universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Machine learning for RNA structure prediction“

verfasst von / submitted by

Julia Wielach BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 875
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Bioinformatik
Betreut von / Supervisor:	ao. Univ.-Prof. Mag. Dr. Christoph Flamm

Acknowledgements

Most of all I would like to thank ao. Univ.-Prof. Mag. Dr. Christoph Flamm for providing the initial idea for my work and supervising my thesis.
I would also like to extend my gratitude to Prof. Dr. Dipl.-Phys. Ivo Hofacker, as my thesis would not have been possible without his support and knowledge.
Special thanks go to Mag. Dr. Michael Thomas Wolfinger as well, for providing assistance and helpful advice.

Thanks to the whole TBI-team, for offering countless opportunities to present and discuss my ideas and helping me along the way. On this occasion I would also like to thank Irene for being an amicable roommate and helping me prepare for presentations.
Among my fellow master students I would especially like to thank Bojana, for always having an open ear for me .

I would also like to express many thanks to my parents, Melitta and Franz, for giving me the opportunity for education and continuous emotional support.
At last I would like to thank my partner Clemens, for keeping me sane during all stress and excitement .
Thank you!

Abstract

Once only thought of as the intermediate step between DNA and proteins, RNA is nowadays known to be involved in various biological processes in a regulating or catalyzing function.
RNA function is usually strongly dependent on its structure, with the structure also being more conserved than the sequence, making RNA structure prediction the subject of research since decades.
RNA folds in a hierarchical way from primary, to secondary, to tertiary and to quaternary structure, forming the secondary structure quite rapidly, while the formation of the tertiary structure is usually a slow process.
RNA secondary structure consists of all canonical base pairs, including wobble base pairs. This includes base pairs of Adenine with Uracil, Cytosine with Guanine and Guanine with Uracil.
Secondary structure is of interest for research, as it is a suitable intermediate step in the prediction of the full tertiary structure and efficient computational methods for its prediction are available.
Traditional methods have been relying on dynamic programming algorithms to find the lowest free energy structure, using nearest neighbor parameters to estimate folding stability.
While these algorithms are efficient and widely used, they have some shortcomings.
For example the restriction on all nested structures, excluding pseudoknots, or the possible neglect of long distance effects.

As Artificial Intelligence methods are gaining popularity in many fields, ranging from image classification to speech recognition and protein structure prediction, they have found their way into RNA secondary structure prediction.
The aim of this thesis is to explore deep learning techniques for their use in RNA secondary structure prediction, to see if they could reach the performance of state-of-the-art dynamic programming approaches or even offer enhancements.
Different network types such as Long-Short Term Memory Networks (LSTMs) or Convolutional Neural Networks (CNNs) are tested as well as different ways to represent input and output.
While several already published machine learning models report good performances, this thesis tries to discuss aspects such as comparability between methods, generalization ability and dataset dependence to offer a broad view on the topic.
Additionally, other measures of performance, such as the representation of global and local secondary structure constraints and the overall topology of structures, are presented.

Zusammenfassung

Einst nur als Zwischenstufe von DNA und Proteinen betrachtet, ist RNA heutzutage dafür bekannt, an verschiedenen biologischen Prozessen in regulierender und katalytischer Funktion beteiligt zu sein.
Die Funktion von RNA hängt üblicherweise stark von deren Struktur ab, wobei die Struktur auch besser konserviert ist als die Sequenz, was dazu geführt hat, dass RNA Struktur seit Jahrzehnten erforscht wird.
RNA faltet sich auf hierarchische Weise von Primär-, zu Sekundär-, zu Tertiär- und schließlich zu Quartärstruktur, wobei die Sekundärstruktur schnell ausgebildet wird, während die Bildung der Tertiärstruktur üblicherweise ein langsamer Prozess ist.
Die RNA Sekundärstruktur setzt sich aus allen kanonischen Basenpaaren,
einschließlich Wobble Basenpaaren, zusammen. Dies schließt Basenpaare von Adenin mit Uracil, Cytosin mit Guanin und Guanin mit Uracil ein.
Die Sekundärstruktur ist von Interesse für die Forschung, da sie eine geeignete
Zwischenstufe in der Vorhersage der Tertiärstruktur ist und effiziente Algorithmen für ihre Vorhersage existieren.
Traditionell wurde dynamische Programmierung genutzt, um die Strukturen mit der niedrigsten freien Energie zu finden, dabei werden Nächste-Nachbar Parameter genutzt, um die Stabilität verschiedener Faltungsmotive abzuschätzen.
Während diese Algorithmen effizient sind und häufig verwendet werden, haben sie dennoch einige Versäumnisse.
Beispiele dafür sind die Beschränkung auf verschachtelte Strukturen, da dies Pseudoknoten ausschließt, oder die mögliche Vernachlässigung von Interaktionen zwischen weit entfernten Basen.

Da die Verwendung künstlicher Intelligenz in vielen Bereichen von Bildklassifizierung zu Spracherkennung und zu Proteinstrukturvorhersage an Beliebtheit gewinnt, haben diese Methoden auch erste Anwendung in der RNA Sekundärstrukturvorhersage gefunden.
Das Ziel dieser Arbeit ist es, die Verwendbarkeit von Deep Learning Methoden für die Vorhersage von RNA Sekundärstrukturen zu überprüfen, um herauszufinden, ob sie eine ähnliche Leistung wie Dynamische Programmierungsalgorithmen erreichen oder auch Verbesserungen bieten können.
Es werden sowohl verschiedene Netzwerktypen wie Long-Short Term Memory Netzwerke (LSTMs) oder Convolutional Neurale Netzwerke (CNNs), als auch verschiedene Arten Input und Output darzustellen getestet.

Während einige, bereits publizierte Methoden gute Leistungen vorweisen, wird in dieser Arbeit auch versucht, Aspekte wie Vergleichbarkeit zwischen Modellen, Generalisierbarkeit und die Abhängigkeit von Daten zu diskutieren, um eine breitere Sicht auf das Thema zu bieten.
Zusätzlich werden alternative Bewertungsmethoden vorgestellt, wie die Berücksichtigung von lokalen und globalen Eigenschaften und die Topologie von Strukturen.

Contents

Part I.

Introduction

1. Aim of this thesis

The aim of this thesis is to gain more insight into the possible benefits, but also the limitations of machine learning approaches used to predict structures for single sequence RNA inputs.
No completely new approach will be developed, however existing approaches will be discussed under new aspects such as generalization ability, dataset dependence and fulfillment of known biological constraints.

The following points will be addressed:

- How well do baseline approaches work?
- Can machine learning offer enhancements to existing approaches?
  What are possible shortcomings in traditional methods which could be enhanced by machine learning approaches?
- Are the datasets used in current approaches suitable to determine performance?
  How should machine learning datasets be chosen to allow comparability and performance evaluation?
- Which other measures of performance could be used to determine, if machine learning model predict feasible structures, based on the current knowledge about RNA folding?

2. RNA biology

2.1. RNA functions and types

Ribonucleic acids (RNAs) are a very important class of macromolecules, controlling or catalyzing many processes related to coding, decoding, regulation and expression of genes. A main characteristic of RNA is that it can both encode information and act as regulator and catalyst.
This versatility of RNA has even inspired the formulation of the RNA world hypothesis, which suggests that the current system with Deoxyribonucleic acid (DNA) for information storage and proteins for enzymatic activity was preceded by an era where both functions were carried out by RNA [Gilbert, 1986, Joyce, 1989, Joyce, 2002].
RNA still acts as information storage, in the form of mRNA, which is the intermediate step between DNA and proteins. Additionally, many viruses use RNA to store their genetic information.
Starting from a scientific worldview were RNA was only seen as the intermediate step between DNA and proteins, more and more non coding RNAs have been discovered. The best known are transfer RNAs (tRNAs) and ribosomal RNAs (rRNAs), both being essential for translation.
Additionally, numerous other classes of non-coding RNAs, with functions in RNA modification and processing, mRNA stability, transcription regulation, gene silencing as well as protein translocation and stability, have been discovered.
It has even been suggested that the complexity of higher organisms requires an additional layer of regulatory non coding RNAs [Mattick, 2003].
This is backed by the fact that an estimated 97–98% of the transcriptional output of the human genome is non-protein-coding RNA as well as organism complexity being generally correlated more with genome size than with the number of genes [Mattick, 2001].

2.2. RNA composition

The monomeric building blocks of RNA are nucleotides, which consist of a pyrimidine or purin base, a phosphate group and a ribose in D-β furanose ring structure.
In contrast to the deoxyribose of DNA, the ribose of RNA has a hydroxyl group attached to the 2' carbon.
This makes RNA less stable, as it is more prone to hydrolysis.
Like in DNA, the bases of RNA are attached to the 1' carbon atom of the ribose and two nucleodides are connected by a phosphate group between the 3' carbon of one nucleoside and the 5' carbon of the next, forming the sugar-phosphate backbone of RNA.

This backbone determines the steric flexibility of RNA, influencing which nucleotides can interact with each other.

The bases found in RNA are the pyrimidine bases Uracil (U) and Cytosine (C) and the purin bases Adenine(A) and Guanine (G), where Uracil replaces the Thymine found in DNA.
In contrast to DNA, with its typical double stranded helical structure, RNA is single stranded. Thus RNA is able to form more complex structures consisting of helical stems, built by intramolecular base pairs, and unpaired loop regions.
These helices are mainly formed by the six canonical base pairs, the Watson-Crick base pairs AU, UA, GC and CG and the Wobble base pairs GU and UG. As CG and GC base pairs form three hydrogen bonds, they are more stable than the AU and UA base pairs which form two hydrogen bonds. The GU or UG Wobble pairs are nearly isomorphic to Watson-Crick base pairs and have comparable thermodynamic stability to AU and UA pairs, forming two hydrogen bonds as well [Varani and McClain, 2000].

About 76% of the total base pairs are of this type , due to the following two fundamental properties [Stombaugh et al., 2009]:
The canonical base pairs are isosteric, meaning that the distances and positions of the C1' atoms are very similar, leading to the possibility to substitute for each other without changing the helical conformation.
The nucleotides in canonical base pairs can share the π orbitals of their atomic rings. This effect is called base stacking and is the major stabilizing factor for helices.

There are several other base pair types found in RNA, all being less stable than the Watson-Crick and Wobble base pairs.
The non-canonical base pairs consist of three main types:
One group has hydrogen bonds between the bases themselves, another group has interactions among the C-H and O/N groups and the last group forms interactions stabilized by polar hydrogen bonds [Hermann and Westhof, 1999].

2.3. RNA folding

In contrast to the fixed double stranded helix structure of DNA, the structure of RNA is greatly variable.
Especially for RNA molecules that act in regulatory functions, the three-dimensional structure is strongly impacting their function.
RNA folding is a hierarchical process [Brion and Westhof, 1997], starting from the primary structure. The primary structure represents the sequence of base pairs from 5'-end to the 3'-end, consisting of the symbols A, G, C and U, representing the bases Adenine, Guanine, Cytosine and Uracil.
In a first step the secondary structure is formed, which consists of the set of base pairs between nucleotides in the RNA sequence.

From this secondary structure the tertiary structure is formed, through bending and folding.
For most RNA molecules the tertiary structure is the final folding level, however some RNAs do associate with other RNAs, forming a quaternary structure
[Jones and Ferré-D'Amaré, 2015].
The number of degrees of freedom in the RNA chain is very high, which makes it difficult to predict the full tertiary structure. However solving the problem of RNA secondary structure is a useful intermediate step, due to the following reasons:
The intrahelical interactions of the secondary structure are responsible for the major part of the free energy of RNA folding.
RNA structure is already conserved on the secondary structure level and can be used to predict function. Due to the hierarchical nature of folding, secondary structure can also be used as a starting point to tertiary structure prediction as well as providing distance constraints.
Additionally, RNA secondary structure prediction can be handled more efficiently by computational methods.

3. RNA secondary structure prediction

3.1. Secondary structure definition

RNA sequences can be described as a string s of length n, using the nucleotide alphabet {A, C, G, U}. A base pair between two nucleotides at position i and j is denoted as (i, j) with $i < j$.
The secondary structure of a RNA sequence is described as a set S of base pairs from the allowed types (A,U), (U,A), (G,C), (C,G), (G,U) and (U,G).
There are two constraints:
The first one being $i - j > 3$, which enforces a minimal hairpin loop size of three nucleotides.
The second one controls that each base is at most part of one base pair, by enforcing $k = l$ if and only if $j = l$ [Waterman and Smith, 1978].
Pseudoknot free structures are fulfilling the constraint that either
$i < j < k < l$ or $i < k < l < j$ is true for any two base pairs $(i, j) \in S$ and $(k, l) \in S$.
As will be discussed later, most secondary structure predictions work by decomposing the structure into simpler substructures, these building blocks are depicted in Figure 1.

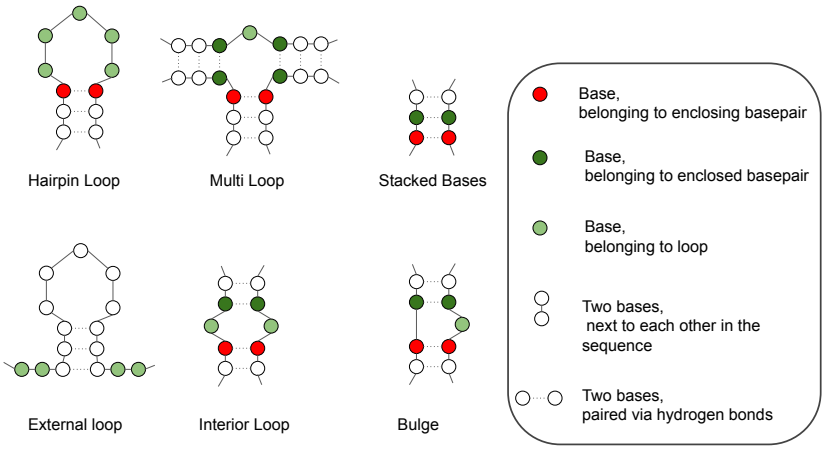

Figure 1.: Different loop-types in RNA secondary structures

Four different loop types are generally distinguished, hairpin loops, interior loops, multi(branch) loops and exterior loops.
They are categorized based on the involved base pairs, with exterior loops being a special case with no involved base pair.

Hairpin loops are loops of degree one, as they only involve the enclosing base pair.
Interior loops are loops of degree two, as they involve the enclosing base pair and one enclosed base pair.
A special case of interior loops are bulges, where one bases of the enclosing base pair (i, j) is right next to one of the bases of the enclosed base pair (k, l), meaning $k = i + 1$ or $l = j - 1$. This forms a loop of size 1.
Stacked base pairs, which are two base pairs (i, j) and (k, l) where $k = i+1$ and $l = j-1$, can be also described as a special case of interior loops of loopsize 0. Several following stacked base pairs are also referred to as a stem.
All other loops, with more than one enclosed base pair, are called multi(branch) loops.
As structures with pseudoknots are are not nested and thus can not be decomposed, their prediction is more difficult and computationally expensive and often not taken into account. Their prediction will also not be part of his thesis.

3.2. Secondary structure representations

As representing a secondary structure as a list of base pairs is not very intuitive for humans, several representations have been developed.
In the following the three representations, which are of interest for this project, will be explained in further detail.

3.2.1. Dot Bracket notation

One common representation method, which is human- as well as machine-readable, is the dot-bracket notation. The structure is represented by a string with the alphabet {".","(",")"}.
A base pair (i, j) is represented by a pair of parentheses at position i and j and unpaired nucleotides are represented as dots.
There are more generalized versions of the dot-bracket notation, which use additional pairs of brackets, such as <>, {}, and [], and matching pairs of uppercase/lowercase letters. This allows them to represent (certain types of) pseudoknots, as base pairs do not have to be nested.

3.2.2. Dot plot

A dot plot is used to represent more information about the folding than just one secondary structure.
In this matrix of $n \times n$, with n being the sequence length, the nucleotide sequence is written on both axes.
The plot is divided in two parts. In the lower left half a dot at the intersection of a column i and row j represents a base pair (i, j) from the RNA secondary structure, for example the minimum free energy structure.
In the upper right half more than one structure is typically visualized.
In this part the probabilities of base pairs are depicted by a dot with a size proportional to the probability of a base pair at this position, for example allowing for the representation of the Boltzmann ensemble.

3.2.3. CT files

In the first line the number of bases and the name of the structure is listed. After that there is one line for every base with the following content:

- Base number (n)
- Base type (A,C,G,U)
- Predecessing Base (n-1)
- Successive Base (n+1)
- Number of the base to which n is paired, alternatively 0 if the base is unpaired

- Natural numbering (often just n repeated)

Multiple structures can be represented by repeating the format for each structure without any blank lines between structures.

3.2.4. Examples

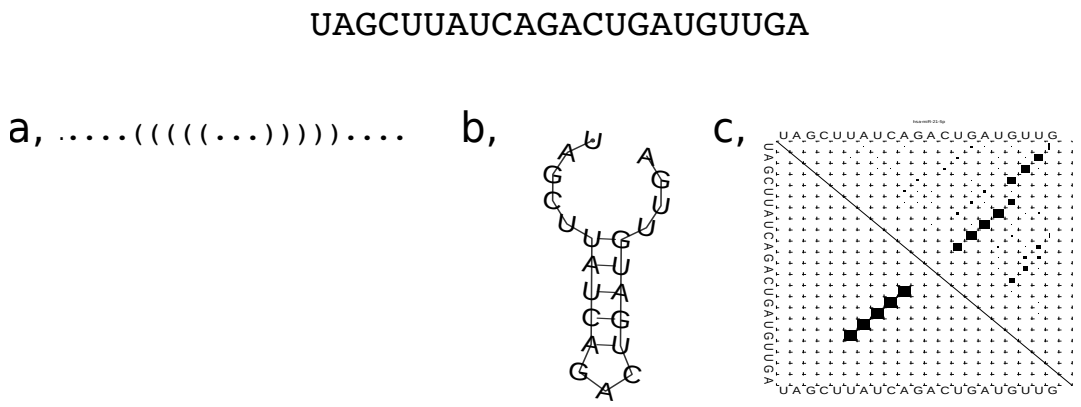

Figure 2.: Three ways of displaying structure information:
A dot bracket structure (a), a secondary structure plot (b) and a dot plot (c). The secondary structure representation is a graphic visualization of RNA secondary structure displaying the sequences and base pairs.
The structure consist of one helix, which can be seen in all representations.
In the dot plot the helix from the MFE structure is displayed in the lower left matrix, in the upper right the probabilities of base pairs within the equilibrium ensemble are visualized.
The MFE structure was predicted with RNAfold and all plots are outputs from RNAfold [Lorenz et al., 2011].
The sequence is Homo sapiens (human) hsa-miR-21-5p, URS000039ED8D_9606. Retrieved January 22, 2021 from https://rnacentral.org/rna/URS000039ED8D/9606.

3.3. Secondary structure prediction algorithms

As the determination of RNA secondary structure by experimental methods, such as X-ray crystallography or NMR, is slow, expensive and technically challenging, research on the computational prediction of secondary structure has been carried out for decades. To enable the computation prediction of RNA secondary structures, it is necessary to evaluate and compare different structures.
Typically this is done by using energy models as scoring functions.
The first and simplest energy model was to favor the structure consisting of the most base pairs. The Nussinov algorithm uses dynamic programming to find the maximum number of base pairs for a RNA sequence [Nussinov and Jacobson, 1980].
While modern secondary structure prediction algorithms still use dynamic programming, the energy model has been refined by using a loop based energy model, which decomposes the structure into loops and assigning energy to these loops based on the nearest neighbor model.
As the name indicates, the nearest neighbor model bases the energy determination on the nearest neighbors of a base, which allows the incorporation of the stabilizing effect that base pair stacking has.
The exact energy parameters are either determined experimentally, with one of the most commonly used parameter-sets being the "Turner" parameters [Mathews et al., 1999, Mathews et al., 2004], or learned from data [Andronescu et al., 2014, Do et al., 2006, Zakov et al., 2011].
Dynamic programming is so widely used, as it solves one major problem in RNA secondary structure prediction, which is that the search space for all valid secondary structures grows exponentially with the sequence length.
This is solved in dynamic programming by assuming a nested structure and iteratively finding the optimal structure by computing the optimal structure for subsequences.

3.4. Prediction accuracy and shortcomings

While dynamic programming models are widely used due to their reliability and computational efficiency, they still have several shortcomings.
On average about 73% of known base pairs are correctly predicted by free energy minimization methods for sequences consisting of fewer than 700 nucleotides, with a lower accuracy for longer sequences [Dowell and Eddy, 2004, Mathews et al., 2004].
At least four major points have been identified to influence prediction accuracy [Mathews and Turner, 2006]:

- Thermodynamic models are incomplete
- Folding kinetics can determine certain secondary structures
- Structure prediction algorithms use approximations, the major one being that all structures are nested

  The restriction on nested structures excludes pseudoknotted structures from predictions.
  Pseudoknots are structural elements with at least two not nested base pairs.
  They make up about 1.4% of base pairs [Mathews and Turner, 2006] and are often over represented in functionally important regions
  [Hajdin et al., 2013, Staple and Butcher, 2005].
  While energy-models that can predict pseudoknots exist, they are computationally intensive and often restricted to certain pseudoknot types.
  Other effects which are not taken into account are triple base pairs and the effect of long distance interactions.

- Some RNAs, such as for example certain riboswitches, may fold into more than one structure

4. Machine learning for RNA secondary structure prediction

4.1. Introduction

Artificial intelligence methods are gaining popularity in many fields, ranging from image classification to speech recognition and protein structure prediction.
Different machine learning methods are used depending on the data to be processed, RNA falls in the category of sequential data, using algorithms originally developed for speech recognition and related tasks.
All algorithms that are used for this work belong to the category of supervised learning and use artificial neural networks, with most of them being deep neural networks.
Supervised learning algorithms learn by using known classifications for the input, such as known secondary structures for input sequences, to iteratively adapt the models predictions.
The RNA secondary structure prediction problem belongs to the subclass of supervised learning called classification. For classification problems the output is restricted to a limited set of values, such as paired and unpaired or opening and closing brackets, representing base pairs.
Lately several different approaches using machine learning methods for the prediction of RNA secondary structure have been published [Chen et al., 2020b, Singh et al., 2019, Zhang et al., 2019].
While these methods report very good performances, they often lack comparability as different datasets are used for training and evaluation.
This thesis aims to be a starting point to gain a better understanding of the capabilities of machine learning for RNA structure prediction and the models that can possibly be used for this task.

4.2. Conceptual formulation and prerequisites

To evaluate the quality of machine learning methods, without having to account for the bias in training data, the first step is to test the methods with results obtained with the existing structure prediction method RNAfold from the Vienna RNApackage [Lorenz et al., 2011]. The minimum free energy (MFE) structure was used.
Sequences are available in much larger variety and quantity than structures and arbitrarily large datasets can be built by using randomly created sequences.

Another important aspect when using machine learning to predict RNA secondary structure is that the result has to satisfy the restrictions for a correct RNA secondary structure (see 2.1 for exact constraints).
Depending on the model this can be different, as different ways to represent output are used.
The dot-bracket notation is often chosen as a standard representation of RNA secondary structure. It uses a sequence of dots and parentheses, with each symbol corresponding to one base in the sequence. Dots represent unpaired bases, while matching brackets represent base pairs.
When using a machine learning method to predict the most likely symbol for each position and then combining them, the results are not guaranteed to satisfy the definition for a correct RNA secondary structure.
One published model used a dynamic programming correction step after the machine learning part [Zhang et al., 2019].
Another approach is to classify all deviations from the constraints as special interactions such as triple base pair interactions, pseudoknots or non canonical base pairs [Singh et al., 2019]. This can be problematic as the data samples, from which such pairs can be learned, are sparse.
A third approach is to enforce these constraints in the training process, this is usually done iteratively [Chen et al., 2020b]. As in this case the iterations are determined based on performance on the dataset that was used for training and evaluation, it is not guaranteed that the exact same method successfully enforces the constraints for other datasets.
The problem can also be avoided by simplifying the RNA structure representation more and trying to predict only paired and unpaired regions. This approach was used for some of the models in this thesis.

One other important characteristic that has to be fulfilled by all models to be used for RNA secondary structure prediction is that they can theoretically be used on arbitrarily long RNA sequences. The length of predicted RNAs is still often restricted as longer sequences need increasingly more computational resources to be included in predictions and training.

5. Machine learning

5.1. Training process

5.1.1. Overview

As mentioned in the previous chapter all algorithms used in this work belong to the category of supervised learning with the subcategory classification.
Furthermore all models are (deep) neural networks using sequential input.

Supervised learning algorithms learn an objective function, which is used to predict the output associated with new input, through iterative optimization. This process runs for several epochs using the training set, which is a subset of the whole dataset.
An epoch is one iteration where the learning algorithm has gone through the entire training dataset.
The number of epochs is typically determined by looking at the performance of the validation set, another subset od the dataset, which is used to test the models generalization capabilities and to adapt the hyperparameters.
Data is split into batches before being fed to the model. A batch is a collection of samples that the model processes before updating the internal model parameters. A sample is one datapoint, such as one RNA sequence.
Optimally the objective function will allow the model to determine the correct output for inputs that were not part of the training data. If that is the case, the model has successfully been able to learn to perform the desired task.
This capability is tested by using a third dataset, the test set. It is especially important, that this dataset has been chosen to be representative of the whole population of samples, that are going to be predicted by the model.

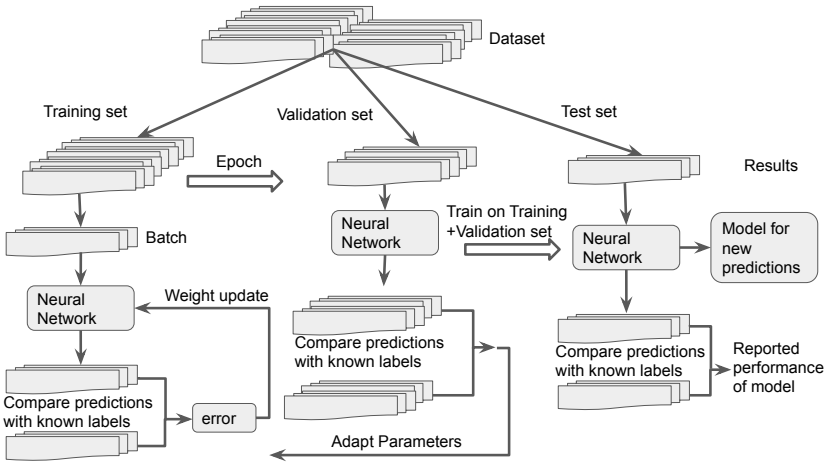

Figure 3.: Overview of the training and validation process

The prediction of correct output for not yet seen samples is referred to as the generalization ability of a model. Two terms are mainly used to describe poor performance in generalization, underfitting and overfitting.
Overfitting is used to describe a model that performs very well on the training set but poorly on unseen data. This happens, if the model learns noise or random fluctuations in the training data and applies the concepts it has learned from these effects on new data. Overfitting can be counteracted by evaluating the training progress against the validation data set and by regulatory mechanisms, such as dropout.
In contrast, underfitting is used to describe a model that performs poorly on the training set as well as on new data. Typically, this indicates that the machine learning algorithm being used is not fitting for the given objective and dataset.

5.1.2. Optimization process

As described, the learning algorithms that will be used in this work learn an objective function. This objective function predicts the output associated with new input through iterative optimization. The most common way to perform this optimization is gradient descent.
"Gradient" in this case refers to an error gradient. A model with a given set of weights and biases is used to make initial predictions and these predictions are then used to calculate an error. The weights are then changed according to these errors, by moving down the gradient (or slope) of error.
In the very first step of learning the calculations are only possible by initializing the weights and biases of the network.
Networks consist of neurons and the weights and biases determine the influence each neuron has on the final output.
The weights determine the strength of the connection between neurons, in other words the weight determines the amount of influence one neuron has.

Biases are constants which allow to shift the activation function. They have no connection to the previous layers, only to the next. Even if the input is zero a bias > 0 guarantees that the activation is not zero.
The activation function determines if a neuron is activated, based on:
$Y = \sum(weight * input) + bias.$

More formally gradient descent is used to minimize the objective function $J(w)$ parameterized by the network model's parameter $w \in R^d$ through updating the parameters in the opposing direction to the gradient of the objective function $\Delta_w J(w)$ in regard to the parameters. The step size with which we reach the (local) minimum is determined by the learning rate [Ruder, 2016].

Three different variants of gradient descent exist, which differ in regard to the amount of data that is used to compute the gradient of the objective function.
The decision of the amount of data to use is a trade-off between the accuracy of the update and the time and computational resources which are needed to perform the update. Additionally this method can not be used for online learning, where new samples are added on the fly.

Batch gradient descent, also called vanilla gradient descent, uses the whole training set to update the gradient estimate once. However, this is extremely slow and can be practically impossible for large datasets or very complex models.

Stochastic gradient descent (SGD) is on the opposite side of the spectrum, as it performs an update of the gradient estimate for each sample. This leads to faster learning and is also suitable for online learning. However, the frequent updates with high variance lead to large fluctuations in the objective function. This can also have the positive effect of allowing the model to escape local minima, while often a reduction in the learning rate is needed to ensure the model settles in a minimum without overshooting.

Mini-batch gradient descent is a compromise between stochastic and batch gradient descent, using a subsample to perform a gradient descent (this is what was defined as batch in the previous section). Typically this is the algorithm of choice, when training neural networks.
In this approach, data is not always presented to the model in the same order, but is shuffled each epoch. This minimizes the risk that batches are not representative of the dataset, which would lead to wrong updates of the gradient estimate.
Several adaptations of mini-batch gradient descent exist, in the following Adam will be described in more detail as it was used in this work. Prior to that, two parameters, used for these adaptations, need to be introduced.
The first one is called momentum, which helps to accelerate the gradient descent in relevant directions and dampens oscillations, as SGD often shows problems navigating along ravines.

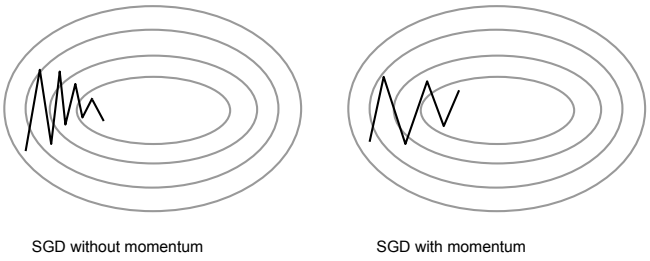

Figure 4.: The gradient descent along ravines of SGD with and without momentum Adapted from "Momentum and Learning Rate Adaptation" by G. B. Orr, 1999. Retrieved January 05, 2021, from https://www.willamette.edu/ gorr/-classes/cs449/momrate.html.

This is done by adding a fraction of the update vector, of the past time step, to the current vector. The fraction is determined by the momentum term, which is typically set to a value similar to 0.9.

The second method is called Nesterov accelerated gradient, which can be seen as a method using momentum with a correction term. It works by first looking at the interim parameters to where the velocity, that was added by momentum, will lead the parameters and correct the update, if the update leads to a bad loss.

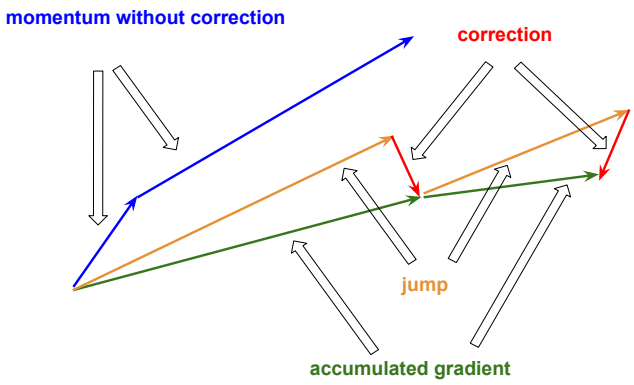

Figure 5.: A comparison of two gradient estimate updates with either standard momentum (blue) or Nesterov accelerated gradient (green after combining red and brown) Adapted from "Neural Networks for Machine Learning, Lecture 6c" by G. Hinton. Retrieved January 05, 2021, from http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Adam is a bit more sophisticated, but simplified it can be described as a combination between momentum and adaptive learning rates. The learning rate in Adam is adapted based on the average first moment (=mean) as well as on the average of the second moment (=uncentered variance) of the gradient.

So far the computation of the gradient has been treated as a black box. The process by which the gradient is called backpropagation, introduced by [Rumelhart et al., 1986]. Essentially backpropagation works by calculation the partial derivative of the cost function for any weight w (or bias b) in the network. This can be used to determine the cost changes that come with changing the weights and biases. The end result of backpropagation is a set of optimal weight, with which the network can then be updated.

The optimization process requires another component which was not yet touched upon, the loss function. As explained, the model navigates along the gradient of the error, calculated by comparing the models predictions to the actual labels. This is also described as optimizing the objective function.
The objective function is often referred to as the loss function and the value calculated by the loss function is simply called the loss.
For classification problems, such as the one presented in this work, a cross-entropy loss is typically used. It is used to determine model weights which minimize the difference between the predicted probability distribution for the dataset and the distribution of probabilities in the training dataset.

5.1.3. Hyperparameters

A machine learning model has weights, which are adapted during training, but also hyperparameters, which are set before training and can not be learned from data.
These parameters belong into two broad categories, firstly parameters which influence model training, such as batch size, number of epochs, learning rate or optimization method and secondly parameters which determine the network structure, such as number of neurons or layers, dropout, weight initialization and activation functions.

We already touched upon batch size and discussed, how it is determined by a trade-off between computation time and gradient update accuracy. A typical default value for batch size is 32, however it is dependent on the network architecture and dataset and values such as 8, 16, 64, 128 or 256 are also often used. All values are power of 2 for efficiency reasons, as CPU and GPU architectures typically organize memory in power of 2.

Batch size is closely linked to learning rate, as the high fluctuations through a small batch size can sometimes be mitigated by using a small learning rate. Choosing a learning rate is a trade-off as well between smooth convergence and the speed of the learning process. Often a decaying learning rate is used, starting with a larger learning rate to enable fast learning and an exploration of the search space in the beginning, while enabling a smooth convergence using a smaller learning rate later.

Several other aspects of the optimization process need to be decided on in the beginning, such as the different optimization variants like mini batch gradient descent with momentum or Nesterov accelerated gradient or Adam. The parameters of these algorithms, such as the value for momentum, are hyperparameters as well.

One last main parameter that influences the network training is the number of epochs, meaning how often the network sees the whole dataset. Usually, the training is stopped when no more increase in validation accuracy is seen.

The general layout of a model is also set via hyperparameters. A model can either be wide or deep, depending if one adds more neurons per layer or more layers.

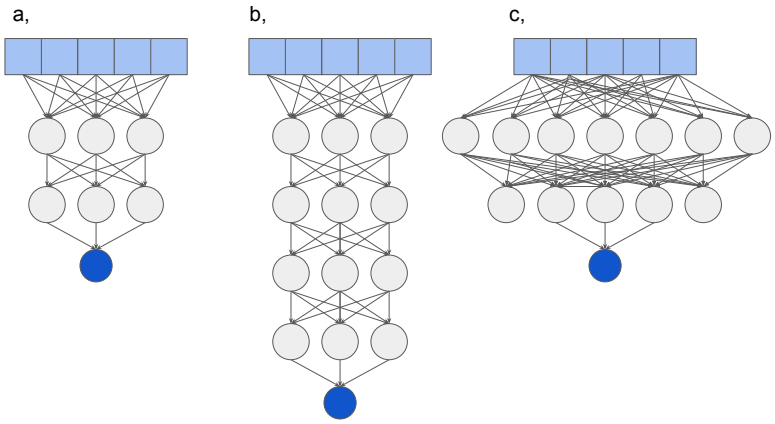

Figure 6.: The capacity of a network (a) can be increased by either adding layers, making it deeper (b), or by adding neurons, making it wider (c).

Dropout is an often used regularization technique. Simplified, it works by canceling the effect of random neurons. The percentage of neurons that are canceled is a hyperparameter.

For training the network weights need to be initialized. Typically, this is done with a method dependent on the activation function.

Activation functions introduce nonlinearity to models, helping deep learning models to learn nonlinear prediction boundaries. They are introduced at the connections of one layer to another layer or to the output. Some activation functions are fixed such as sigmoid in the output layer for binary classifications and softmax in the output layer for multi-class classifications.
The sigmoid function output values between 0 and 1, based on the function $\sigma(x) = \frac{1}{1+e^(-1)}$.

For intermediate connections the tanh functions has been predominantly used,
however it is more and more replaced by the Rectified linear units (ReLUs) and their adaptions.

Setting hyperparameters is one of the main difficulties of every machine learning problem, as it requires either expert knowledge or vast computational resources.
One approach is to set these parameters according to knowledge from past experiments, together with literature recommendations and non-systematic tests.
Several alternatives. such as grid search, random search or bayesian optimization exist, which require large computational and time resources.

Grid search tests all combinations of different hyperparameters, which is often not feasible for large models or datasets or for trying new combinations of values for parameters. Usually, this method is used to test a limited set of parameters, that are known to perform well.
Random search randomly samples combinations of hyperparameters and is used to try new combinations or values, however it is not guaranteed to find the optimal solution and can take some time.
The third, more sophisticated approach is bayesian optimization, which works by building a probability model of the objective function and using it to select the most promising hyperparameters.

5.2. Network types

5.2.1. Sliding-Window Method

The sliding-window method is a simple way to process sequential data, as it converts a sequential supervised learning problem into a classical supervised learning problem.
It works by converting the input sequence into a set of windows, one for each symbol in the sequence, consisting of a central element and the right and left context symbols. Each of these windows is mapped to an individual output value and all the output values are concatenated to form the final sequential output.
The advantage of this method is that any feed forward network can then be applied, for example a simple feed forward network or a convolutional network. A feed-forward network consists of multiple layers having connections in only one direction. A simple example are multilayer perceptrons, where all neurons in one layer are connected to all neurons in the next layer. When all neurons from the prevoius layer are connected with all neurons from the current layer the current layer is also called fully connected (FC).
Convolutional neural networks are an adaption of multilayer perceptrons, which work by generating more complex patterns from simpler and smaller patters. Convolution networks are often used on multidimensional data, which will be later discussed in more detail, however there are also variants which can be used on sequential data.

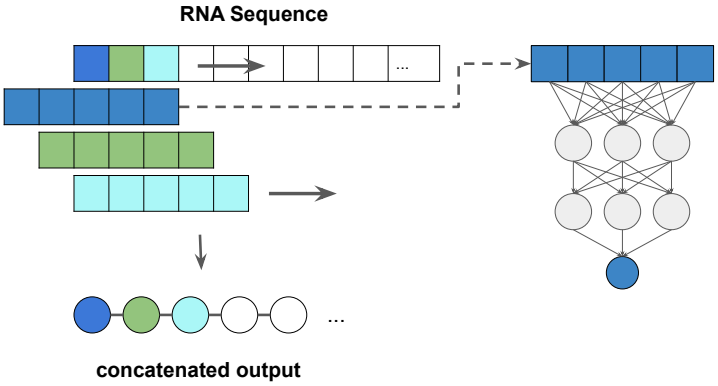

Figure 7.: An overview over the sliding-window algorithm with a simple feed forward network and a window size of 5

Nonetheless it suffers from several shortcomings, such as choosing the window size and neglecting certain correlations in the data, for example interactions across longer distances as the window size.

5.2.2. Recurrent Models

Recurrent Neural Networks(RNNs)

One class of neural networks, which are naturally able to support sequential data, are recurrent neural networks (RNNs).
RNNs are recurrent in the sense that they perform the same task for each element in a sequence, with the output being dependent on previous events. This also allows efficient computations, as the same parameters can be shared for the whole sequence, allowing the models to process sequences of arbitrary length.
RNNs have loops in them which allow the information to persist. This is depicted in Figure 8, which also shows the unrolled version.

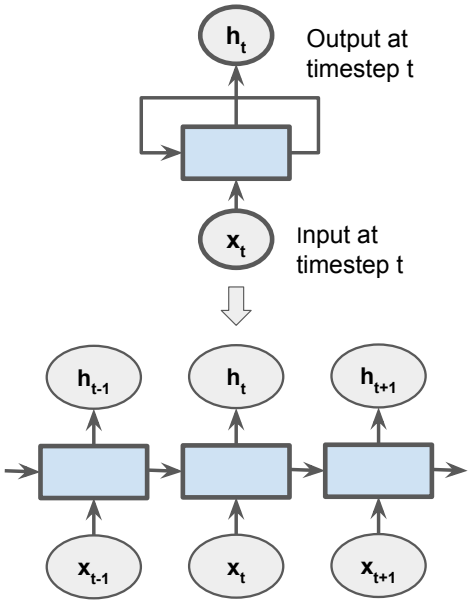

Figure 8.: The structure of RNNs in loop and unrolled version- h_t is the output a time t, while x_t is the input at time t.
Adapted from "Understanding LSTM Networks" by C. Olah, 2015, colah's blog. Retrieved January 05, 2021, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

One problem of RNNs is that they have problems learning long-term dependencies, at least in practice. As there are fundamental difficulties [Bengio et al., 1994], another approach is needed.

Long Short Term Memory Neural Networks (LSTMs)

Long Short Term Memory networks (LSTMs), introduced by
[Hochreiter and Schmidhuber, 1997], are the solution for processing data with long-term dependencies. They are a variant of RNNs and have essentially the same loop-type architecture.
However, other than the simple repeating module used in RNNs, see Figure 9, their repeating module has four different components, whose special interactions enable a proper long-term memory, see Figure 10.

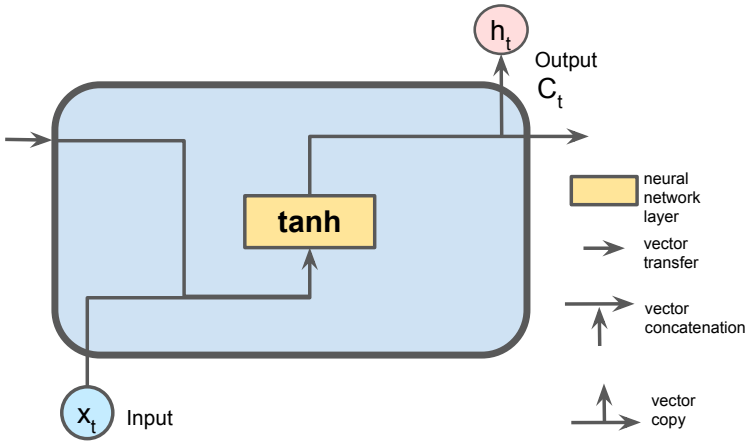

Figure 9.: An schematic view of the repeating module in RNNs
Adapted from "Understanding LSTM Networks" by C. Olah, 2015, colah's blog. Retrieved January 05, 2021, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

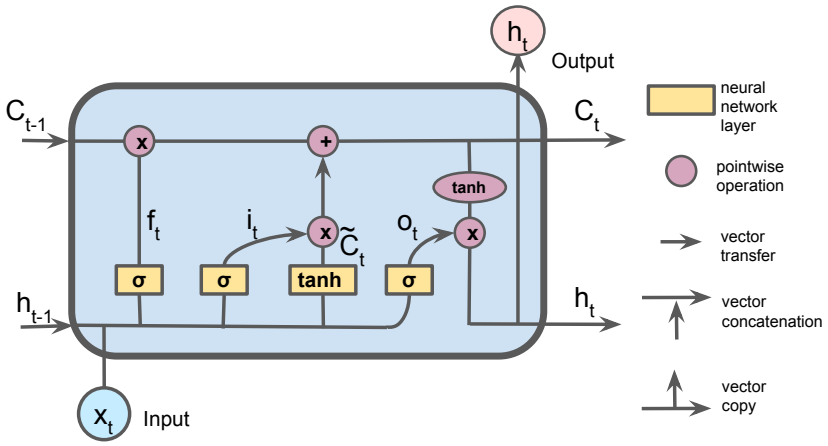

Figure 10.: A schematic view of the repeating module in LSTMs
Adapted from "Understanding LSTM Networks" by C. Olah, 2015, colah's blog. Retrieved January 05, 2021, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

The main mechanism which allows a long-term memory in LSTMs is the cell state C, which is the horizontal line on top in Figure 10.
Information in this cell state can only be changed, updated or removed by regulated interfaces, the gates.

The first gate is the forget gate f_t. This gate decides which information in the cell state will be forgotten. The sigmoid layer uses the output from the previous time step and outputs a number between 0 and 1 for each number in the cell state, with 0 meaning complete deletion and 1 complete retention.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f]$$

The decision about what will be stored in the cell state is regulated in two steps.
The input gate decides on the values that will be updated. Additionally a tanh layer creates new candidate values for these positions. These layers are then combined to update the cell state.

$$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i]$$

$$\widetilde{C}_t = tanh(W_C \times [h_{t-1}, x_t] + b_C]$$

Now we can compute the new cell state:

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C}_t$$

One last gate, the output gate o_t, decides which part of the cell state the model will use as output. First a sigmoid layer decides in the parts of the cell state that will be part of the output. Then a tanh layer pushes the values between -1 and 1 to multiply them with the sigmoid gate to create the output

$$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o]$$

$$h_t = o_t \times tanh(C_t)$$

W and b in the formulas are the weights and biases, which are the trainable parameters of the LSTM.

Several variants exist, such as adding peepholes, which allow the gate to not only look at the output of the last step and the new input but also at the cell state.
Another variant combines the input and forget gates.
A more dramatic change is done in Gated Recurrent Units (GRUs), where forget and input gates as well as the cell state and hidden state are combined along with some other minor changes, resulting in a simpler architecture.
While some of these variants, such as GRUs, are gaining popularity, studies suggest that there might be only minor performance differences among those variants [Greff et al., 2015].
For all variants the last layer is a fully connected one, which maps the extracted properties into a final output. For binary classification this layer uses a sigmoid activation to output values between 0 and 1.

Bidirectional Networks

One variant of LSTMs, which is useful for this work, is the bidirectional architecture [Schuster and Paliwal, 1997].
Both RNNs and LSTMs can be designed in this way, which is based on the idea that both previous and future elements are important for the output at a specific time step. This is the case for RNA sequences, as base pairs can be formed with bases on each side on the currently processed base pair. It is also of interest how far from both ends of the sequence a base is located.
The bidirectional structure is pretty simple with two layers stacked on top of each other, each processing the output in a different direction, see Figure 11.
The output of both models is then combined, with the default method being concatenation.

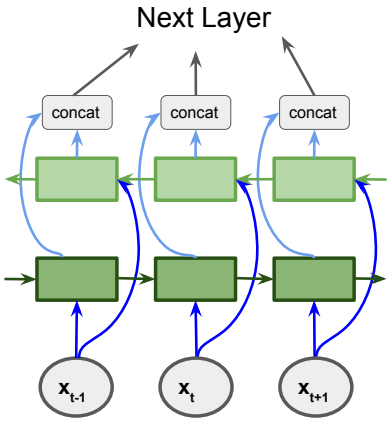

Figure 11.: A schematic view of the bidirectional architecture for recurrent networks.

Additionally, layers can then either be bidirectional or only one directional, as information of both directions is already included by the first layer.

Multidimensional Networks

Another variant of the recurrent architecture has been developed to be able to use the benefits of the recurrent architecture for multidimensional input [Graves et al., 2007].
The basic idea of multidimensional recurrent network (MDRNNs) is to replace the connections used in standard RNNs with as many recurrent connections as there are dimensions. In Figure 12 a schematic view of the 2D case can be seen. where the position (i,j) gets input from two directions. Similar to the bidirectional architecture, multiple layers are needed to provide the whole context, as can bee seen in Figure 13.

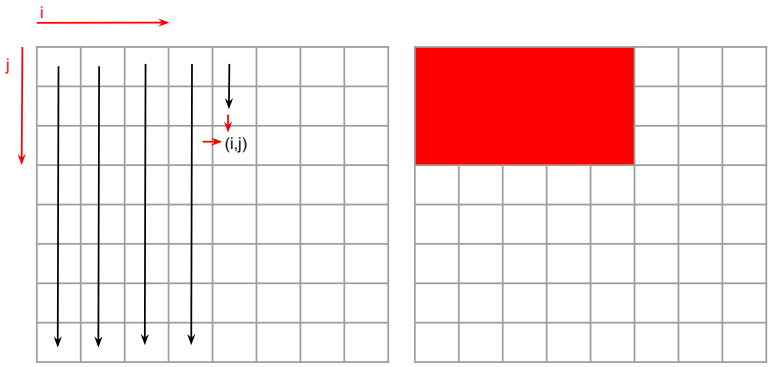

Figure 12.: One layer of a network in 2D getting information from directions i and j. The red area marks the context, which is available to the model at a given point (i,j)

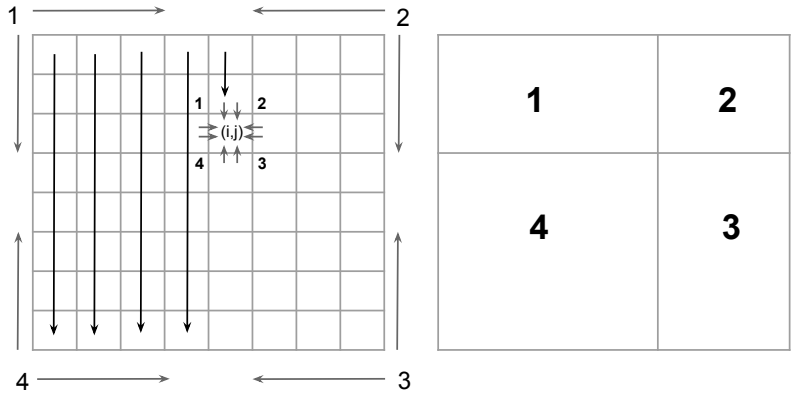

Figure 13.: A schematic overview of four layers which provide the whole 2D overview at a point (i,j).

More formally 2^n separate hidden layers are needed to extend BRNNS to n-dimensional data, where the axes are chosen in such a way, that their origins can be found on the 2^n vertices of the sequence.

Another simpler approach to use RNNs or LSTM for two dimensional data is to use four layers and let them process the input in a sequential way in all four directions, see Figure 14 [Visin et al., 2015]. At the end the output of all layers is combined. One method to do this is concatenation like for the bidirectional architecture.

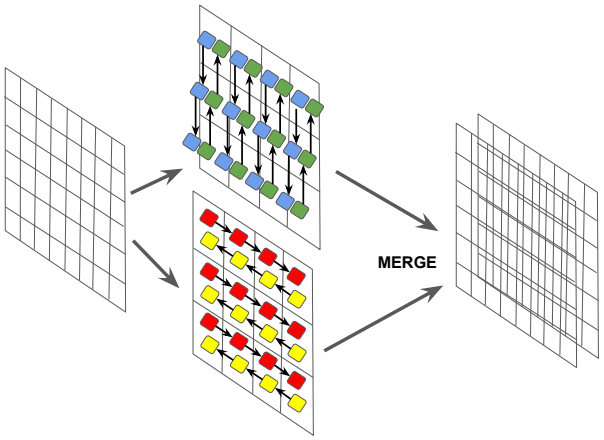

Figure 14.: Two sets of bidirectional networks (green&blue and red&yellow), which provide the whole context information of a 2D input.

5.2.3. Convolutional Neural Networks (CNNs)

Another type of networks which can process multidimensional input are Convolutional Neural Networks (CNNs). Typically, they are used for images, however they can be also applied to other data of the same dimensionality.
CNNs are designed to learn spatial hierarchies of features, starting with low level hierarchies and building up to high-level patterns.

Usually three types of operations are performed:
Convolution steps, where kernels slide along the data to compute output for the position in the middle of the kernel based on the trained parameters of the kernel and the surrounding values. In Figure 15 one can see how this is done, by first computing the element-wise product with the kernel and then summing up the values to get the output.
As the same kernel parameters are used across the whole input, arbitrary input sizes can be processed. Multiple of these kernels are used to extract different features with multiple convolution layers building up small patterns to larger more complex ones.
The size of the kernels, which are typically 3x3, 5x5 or 7x7 and the number of kernels are hyperparameters of the model, the only learned parameters are the parameters of the kernels. Another step is the pooling, which is a downsampling operation and as such not used in this work, as input and output are supposed to be of the same size.
The last layer is again a fully connected one, which maps the extracted properties onto a final output. For binary classification this layer uses a sigmoid activation to output values between 0 and 1.

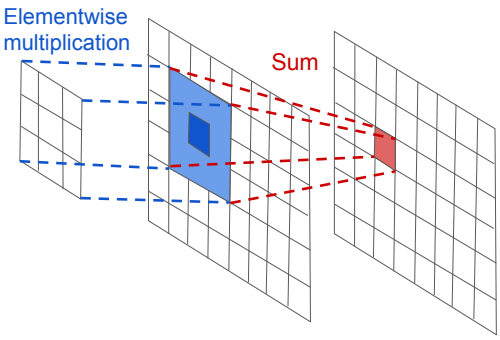

Figure 15.: Here one kernel of size 3x3 (blue) is shown, currently producing output (red) for the position in dark blue. This is done by elementwise multiplication of the kernel with the input and summing up the result.

Although CNNs depend on multiple layers to build up more complex patterns, there is a maximum number of layers and performance actually decreases after a certain point.
One solution that allows CNNs to get deeper and actually increase their performance thereby are Residual Networks (ResNets) [He et al., 2016].
ResNets consist of building blocks such as depicted in Figure 16, which introduce skip connections to the networks.

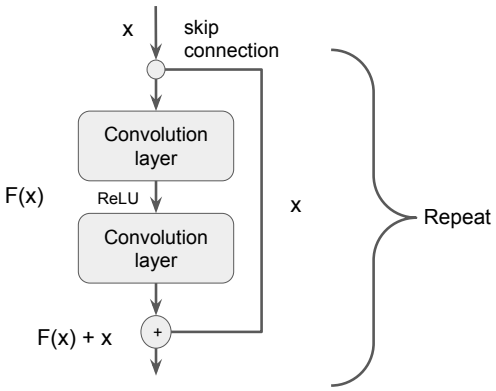

Figure 16.: One building block of a ResNet.
Adapted from [He et al., 2016].

Part II.

Methods

6. Datasets

6.1. Data generation

One major shortcoming of machine learning is the dependence on large, high quality datasets.
RNA structure datasets are usually biased to certain types of RNAs, such as rRNAs. This bias is caused as some RNAs, such as those being in the focus of research or those easier to isolate and crystallize, are much more prevalent. Usually these datasets are also quite limited in the number of RNA sequences, especially those of high quality.

Additionally, especially in the field of RNA research, the usage of different datasets does not allow meaningful comparisons between the reported performances of different machine learning models.
This is especially true for machine learning
approaches, as their reported performance can greatly vary dependent on the datasets used for training, validation and tests.
Performance differences are still observable for non-learning dynamic programming
approaches , however they are more predictable such as a poorer performance for longer sequences.

One important consideration when constructing and choosing datasets for machine learning is that the models need to be validated and tested on datasets which are dissimilar enough from the training set.
For machine learning too similar sequences are usually removed from datasets using sequence similarity measures [Chen et al., 2020b, Singh et al., 2019].
While the authors in [Chen et al., 2020b] perform a permutation test to show that their training, validation and test set are different enough to claim a good generalization ability, their test set is restricted to the same types of RNAs used in the training and validation sets.
In [Singh et al., 2019] only one of the test sets, which is very small with only 67 sequences, is constructed using a filtering based on both sequence similarity and homology.
Different groups of RNAs are not at all discussed in this publication.
A completely different approach was taken in [Zhang et al., 2019] where results are derived from training and testing the networks on only one type of RNA at a time, making it difficult to compare to other, more generalizable approaches.

It has been shown that differences on the sequence level are not enough to detect overfitting and tests with structurally different RNAs need to be performed [Rivas et al., 2012, Rivas, 2013].
It has also been shown that a certain amount of structural diversity is necessary to train RNA secondary prediction models, especially with a large set of parameters.
Another point to take into consideration is that more complex models generally require larger training sets.

To be able to compare models and their performances, without having to take the probable bias and limited availability of data into account, artificially created data was used.
This also enables the generation of arbitrarily large datasets.
RNA sequences were created randomly, using no additional constraints such as selecting sequences representative for different RNA types or using specified GC contents.
The secondary structure, or more precisely the MFE structure, for these sequences was then predicted using RNAfold of the ViennaRNA package [Lorenz et al., 2011].
As a starting point a length of 70 nucleotides per RNA sequence was chosen to enable fast training and implementation and better comparability. Two different sizes of training sets were used, 80000 or 8000 sequences, with validation sets consisting of 20000 or 2000 sequences respectively.
Originally tests of datasets with sequences up to 500 or 250 were planned, however this was not possible due to GPU restrictions.
Instead four datasets with different distributions, see Figure 17, were used. The datasets consist of sequences with lengths ranging from 25 to 100. The training sets consist of 30000 sequences and the validation sets of 5000 sequences with approximately the same distribution.
To test different sequence lengths datasets with lengths from 30 to 250 consisting of 2000 sequences for each length were used as test sets.

Figure 17.: The four different distributions.
The plots are based on the training sets, however the validations sets have the same overall distributions.

6.2. Data preparation

Due to the nature of machine learning models, data has to be presented to them in a special encoding.
The models can only use numerical input and not, for example, symbols, such as A, G, C and U for bases. Thus a numerical equivalent for each base has to be chosen.
Typically, this is done by using one-hot encoding, which transforms a categorical feature, for example the bases, into a numerical feature. The final numerical feature can then be used as input for a machine learning model.
A simple integer label would not be sufficient, as it conveys a natural order in the data which is not present with the original categorical features. For example, if the bases A, C, G and U are represented by 1, 2, 3 and 4, the model could now see G as the average of C and U ((2+4)/2=3).
One hot encoding is a binary representation using a vector with N fields to encode N different values, with the all fields being 0 besides the n^{th} field being 1 to represent the n^{th} value. The encodings we use in this case are [1,0,0,0] for A, [0,1,0,0] for C, [0,0,1,0] for G and [0,0,0,1] for U. For certain methods which will be described later the input is a matrix of one hot encodings.

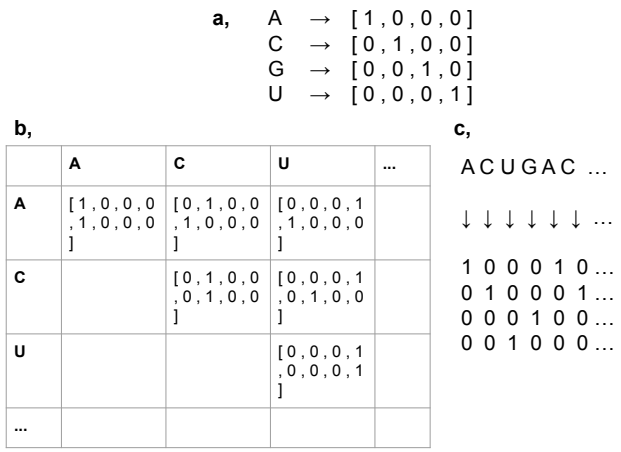

Figure 18.: The one hot encoding for individual bases (a) can either be used to represent an input matrix (b), by concatenating the corresponding one-hot encodings for each position, or to represent a sequence input(c)

The encoding which is chosen for the labels is also important, as it determines the kind of output a model will produce. As discussed in the introduction, predicting the full secondary structure with a machine learning model is difficult due to needing an additional step to make sure that all the predicted symbols together represent a correct secondary structure.
For example, when using the dot-bracket notation, a correction step needs to ensure the same number of opening and closing brackets.
As a first step the structure was simplified by classifying each base as either unpaired or paired, depending on the dot bracket notation predicted by RNAfold.
In this case, a string consisting of 1s and 0s representing paired and unpaired regions of the sequence was used as label for the input. The second approach was using a matrix to encode the likelihood of each nucleotide in the sequence to form a base pairs with any other nucleotide in the sequences.
In this case the label used is a matrix with 0 at position (i,j), if base i and base j are unpaired and with 1 at position (i,j), if base i and j are paired.

Another important property all machine learning models have to possess to be suitable for RNA structure prediction is the ability to use different sequence lengths as input. The models presented in this paper fulfill this restriction, as explained in chapter 4.4.
To enable a correct training when using datasets with different sequence lengths two methods called padding and masking have to be used.
Padding brings all sequences to the same length by adding trailing or leading characters.

Very often a method called zero padding is used, which, as the name indicates, adds zeros to bring all sequences to the same length.
This process is needed as tensors used in tensorflow and keras have to be of the same size along each axis, with the exception of ragged and sparse tensors.
For all input in this work a padding value of [0,0,0,0] was used, with padding being used for all models of the ResNet and 2D-BLSTM Type.
For the sliding window method padding is used for the start and end of sequences, where the sequence does not fill the full sliding window.
In this case the next step called masking is not done, as it is valuable information for the prediction, if a base is at the edges or the middle of a sequence. For the LSTM used on unpaired/paired input no padding has been used, as it was only trained on single length datasets.
A second step is needed to allow the model to distinguish between input that carries information and padding, this is done by masking. Masking provides the model with an additional input of the same size as the sequence with padding. This additional input is 0 where sequences was padded and 1 otherwise. This leads to the influence of the padded part being canceled while training the model.
A similar process is used to treat the edges of convolutional networks. Without edge values a convolutional network would lead to a decrease in datapoints, which is problematic for RNA as we want one datapoint per base or base pair.
In this case edge values are used, to allow the model to keep the same dimensions for all layers. Again zero padding has been used for all work presented in this thesis.
Typically, convolutional networks are used for image processing where reflection padding is widely used. However, zero padding or padding with a specific value is more applicable for RNA structure prediction, as the location within the sequence influence the likeliness of a base being paired or unpaired, as mentioned above.
Due to computational restrictions a method called bucketing can be performed before using padding and masking. Input for machine learning models has to be only of the same size for one batch, which is a subset of data presented to the model before one gradient update is done.
As the computational resources and time needed to train the model rise with the length of the sequence, it can decrease the time and resources needed, if sequences are only padded to match the longest sequence in their respective batch and not to the longest sequence in the whole dataset.
To use bucketing most efficiently the dataset has to be sorted before being split into batches. This has the downside of making shuffling between different epochs less flexible, however it is often needed when time or resources are limited such as for this thesis.

7. Implementation

7.1. Framework

Networks were implemented using Keras version 2.4.0 with Tensorflow version 2.3.0.
Training was either performed on the CPU cluster of the Theoretical Biochemistry Group at the Institute for Theoretical Chemistry in Vienna or using the GPU Google Colab provides with a free membership.
As access to the free GPU provided by Google Colab is restricted, such as by a time limit of continuous usage and a restriction of usage after intensive computations, GPU usage was a limiting factor.
Additionally, the available GPU memory was also a restricting factor. Some models could only be trained with a certain maximum batch size, for example batch size 16 had to be used for the ResNets (model 1 in Table 1) on the datasets with sequence length 25-100. Other models could not be trained on certain datasets at all, like the ResNet+2D-BLSTM(model 3 in Table 1) on the datasets with sequence length 25-100.
However, as the computational times in Figure 19 indicate, the training of the ResNet+2D-LSTM and ResNet models would not have been possible at all without a GPU in the time range of this thesis.

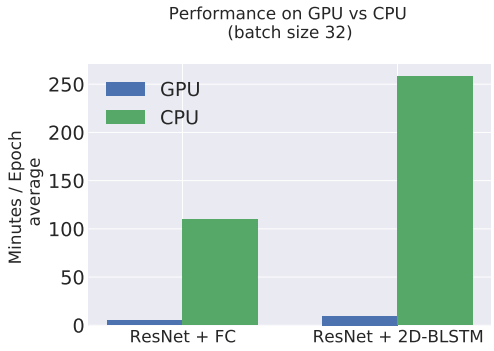

Figure 19.: Run times for the ResNet+FC layers and ResNet+ 2D-BLSTM architecture, for exact parameters, see model 0 and model 3 in chapter 6.2 respectively.
Models were trained on 8000 sequences of length 70, the run times are averages of 5 epochs.
Data was fed to the model via generators and no multiprocessing was used.

Both the sliding-window approach and the bidirectional LSTM can be run on only CPU in a reasonable time.

7.2. Network architectures and parameters

In the following, the architectures and parameters of all models as well as the choice of models will be described.
Additionally, the exact datasets used for training will be described.

7.2.1. Models with sequence output

As described in chapter 4.2 and chapter 6.2, for machine learning models which predict the most likely symbol for each position and then combine them, the results are not guaranteed to satisfy the definition for a correct RNA secondary structure.
An easy way to avoid this is to simplify the RNA structure representation by predicting only paired and unpaired regions. In this case, a string consisting of 1s and 0s representing paired and unpaired regions of the sequence was used as output of the model.
This approach was used for the first two network types, which were implemented: feed forward networks using a sliding-window method to process variable length data and bidirectional long short term memory networks (B-LSTMs).
For machine learning projects it is useful to have a simple baseline model to be able to evaluate the performance of more complex models. In this case the sliding-window method was chosen, which converts a sequential supervised learning problem into a classical supervised learning problem.
The sliding-window models were trained with a batch size of 128 for 100 epochs for every dataset.

Optimization was done with the Adam optimizer using standard settings.
Two different architectures were used. A simple, completely connected feed forward network and a more complex 1D convolutional network, see Figure 20 and 21 respectively.
The convolutional network was inspired by models used for protein structure prediction [Angioloni, 2019].

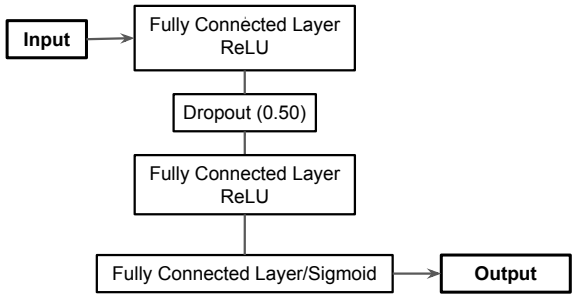

Figure 20.: A schematic view of the simple feed forward architecture.
The first fully connected layer consists of 128 neurons, the second one of 32.

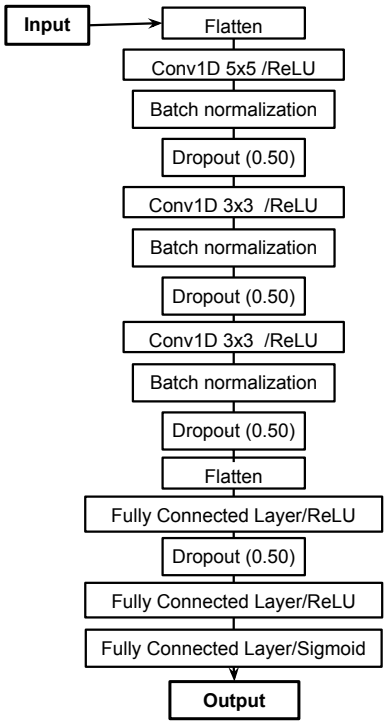

Figure 21.: A schematic view of the more complex 1D convolutional architecture. The first and second convolution layer use 128 kernels, while the third one uses 64. As in Figure 20 the first fully connected layer consists of 128 neurons, the second one of 32.

After some initial tests the window sizes 15, 35 and 71 were chosen to compare different context fields for the models. With the largest window-size 71 the model was able to "see" the whole sequence of length 70 as context.

LSTMs were implemented next, as they are naturally able to support sequential data and are capable of taking long term dependencies into account. The bidirectional approach has been chosen, as context from both sides of the sequences is necessary to make predictions about the pairing of a given base. The BLSTM networks were trained with a batch size of 128 for 100 epochs for every dataset as well.
Optimization was also done with the Adam Optimizer using standard settings.
Adam was chosen as initial tests with SGD were unstable, terminating with NaN errors. The same was seen with both Adam and SGD with deeper or wider networks, thus pretty simple networks were used.
Three different networks were used, two with one bidirectional layer and 40 or 80 neurons respectively and one with three bidirectional layers with 40 neurons.
Other layouts such as adding additional LSTM and Fully Connected layers were planned, however as several tests terminated with NAN errors again, this was not pursued further.

For both the sliding-window method and the LSTM learning curves were produced to evaluate, if more training data could further improve performance. Learning curves use subsets of the training set, while the validation set stays the same. A training set of 80000 sequences and a validation set of 20000 sequences all of length 70 were used as basis, with subsets of 10, 100, 1000, 10000, 25000, 50000 and 80000 sequences. The training on each subset was stopped after 100 epochs.
To evaluate the variability in results between different runs, all models have been trained twenty times using the same dataset of sequences with length 70, with 8000 sequences for training and 2000 for validation.
In addition all models have also been trained using twenty different datasets all with sequences of length 70, with 8000 sequences for training and 2000 for validation.

7.2.2. Published methods

Several other methods were recently published, so the next approach was to chose one of these approaches for further evaluations.
The earliest published method was CDPfold [Zhang et al., 2019], a convolutional network approach based on a scoring matrix as input (see Figure 22). It relies on a dynamic-programming correction step at the end to predict secondary structure in the dot bracket notation.

	A	C	G	U
A	0	0	0	2 +3 * b
C		0	3 +2 * b	0
G			0	0.8
U				0

Figure 22.: The scoring matrix used as input for CDPfold.
CG/GC pairs get a score of 3, AU/UA a score of two and GU/UG pairs a score of 0.8.
An additional factor is added, if base pairs are right next to each other on the same diagonal. This rewards stacked base pairs, which are known to stabilize secondary structures.

The part of the matrix, which is used as input for the convolutional network is chosen in regard to the maximum stem length in the training set.
This combined with the fact that the results presented were derived from training and testing the networks on only one type of RNA at a time, make the approach difficult to compare to other, more generalizable approaches.

The next recently published architecture is an approach combining a ResNet with a 2D-BLSTM, such as in Figure 23, called SPOT-RNA [Singh et al., 2019].

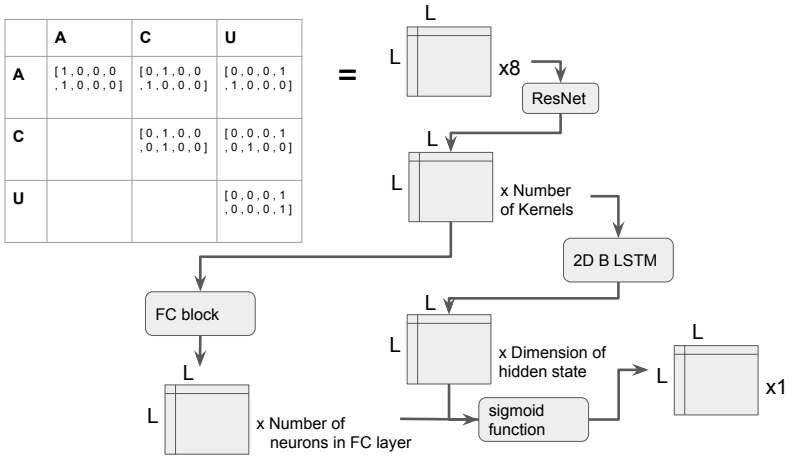

Figure 23.: An overview of the SPOT-RNA architecture.
A matrix of concatenated one hot encodings is used as input for a ResNet, which is followed by a 2D-BLSTM or a block of fully connected (FC) layers. In the publication five different variations of this model are used to build an ensemble for prediction. Thus several variation of this networks exist, with one main difference being the usage of either the LSTM or FC block.
The output is a matrix with a binary value representing the probability of a pairing at that position.

The model uses a matrix of concatenated one-hot encodings as input.
The output is a NxN upper triangular matrix, where N is the RNA sequence length. The matrix consists of values between 0 and 1 representing the likelihood of each nucleotide pairing with any other nucleotide.
A threshold is used to determine if two nucleotides are pairing. The threshold is chosen after optimizing for performance on the validation set. The threshold value used in SPOT-RNA is 0.335.
This approach does not guarantee all constraints which are generally used for RNA secondary structures are fulfilled. The first point is that pseudoknots can be included. The other one is that base pairs are not exclusive, so if (i,j) are predicted as pairing it is not guaranteed that the bases i and j are not in other base pairs.

This is treated as feature by the authors of SPOT-RNA. However from the limited amount of data available for these structural features it is not clear, how well the model is able to represent them. Another published method, called E2Efold [Chen et al., 2020b], uses two coupled networks, first a transformer-based deep model, which learns the sequence information used for structure prediction and second a multilayer network, which enforces constraints in a gradual way and restricts the output space to valid secondary structures. However in this approach it is difficult to scale the same network for training other datasets. The multilayer network, which enforces constraints, is optimized iteratively and stopped when good performance is reached for the used dataset. When other datasets are used, the process is possibly stopped too early, leading to an insufficient enforcement of constraints and the prediction of invalid structures.
As the training sets in this thesis are restricted by computational resources and are thus very different from the original datasets, it was not possible to use these networks in a meaningful way.

7.2.3. Model with matrix output

From the three methods discussed in the last chapter SPOT-RNA was chosen for further tests, as it was trainable with the computational resources available in constraints to E2Efold while also being more generalizable than CDPfold.
As the exact code for the original models architecture is not available an implementation was based on the description provided in the paper as well as on the graphs of the pre-trained models.
The input is a matrix of concatenated one hot encodings, as shown in Figure 18.
Similar SPOT-RNA the output is a matrix of values between 0 and 1 representing the likelihood for each nucleotide pairing with each other nucleotide.
The output on which the first performance reports are based is the full NxN matric, the exact limitations of using the triangular matrix are discussed in the next chapter.
The exact network structure can be seen in Figure 24 and the exact hyperparameters are listed in Table 1.
The exact method used for a multidimensional LSTM (referred to as 2D-BLSTM in the rest of this thesis) is implemented as described in chapter 5.2.2., Figure 14. Two bidirectional LSTM networks are used, processing the matrix vertically and horizontally.

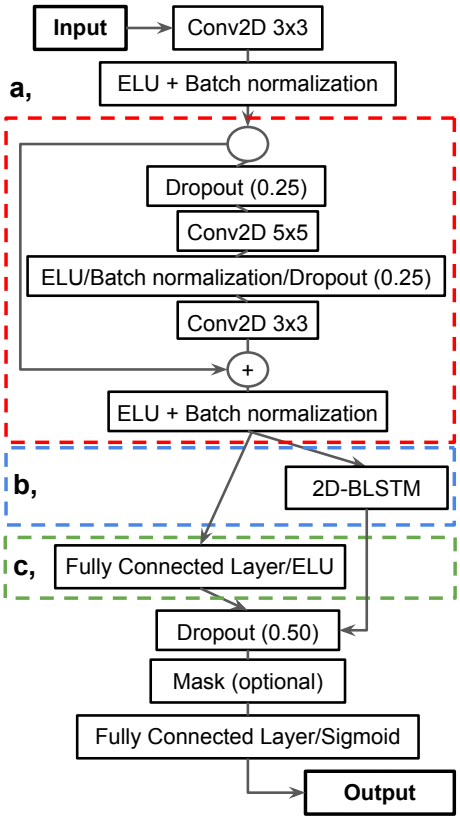

Figure 24.: A more detailed view of the network structure.
The exact repeat numbers for the ResNet (in red), the LSTM (in blue) and the fully connected layers (in green) are listed in Table 1.

Model	Filters	ResNet blocks	LSTM blocks	LSTM neurons	FC blocks	FC neurons
0	64	20	0	0	2	512
1	48	16	0	0	2	512
3	64	30	1	200	0	0

Table 1.: The exact parameters for the models used in this thesis, the values are taken from the original SPOT-RNA publication.
The number of filters in the initial 3x3 convolutional layer was set equivalent to the number of filters in the ResNet blocks, as this was not specified in the original publication.

For training the optimizer Adam was used with standard settings, such as in the original work.
Initially models were trained on the same dataset as the sliding-window approach and the LSTM with length 70 and a training set of 8000 sequences with a test set of 2000 sequences.
Model 3 and 0 were trained with this dataset to get a comparison between using either the 2D-BLSTM or the FC block. The maximum possible batch size for each network was used for training which was 64 for model 0 (ResNet +FC) and 32 for model 3 (ResNet + 2D-BLSTM). Model 1 was also trained with this dataset, as it was used for the following test, a batch size of 64 was used. Training was stopped after 50 epochs.
Again learning curves were done, this time using the smaller dataset of 8000 sequences for training and 2000 for validation, with subsets of 100, 1000 and 4000 sequences. The training on each subset was stopped after 50 epochs.
The length of sequences and the distributions of these length in the training set is one major difference between datasets, whose effect can be tested within the constraints of this thesis.
The datasets which were used can be seen in Figure 17. Training sets of 30000 sequences and validation sets of 5000 sequences were used. Due to computational constraints only model 1 could be used for the tests with the differently distributed datasets. A batch size of 16 was used and training was stopped after 35 epochs.
Performances for different sequences lengths were also tested, by using datasets consisting of 2000 sequences of one length. Lengths from 30 to 250 in steps of 10 were used.

8. Methods for performance evaluation

8.1. Metrics

The cross entropy loss also called log loss is used to train the model, as it is the standard for binary classification problems. Log loss especially penalizes predictions which are confident and wrong.

For binary classifications the log loss can be calculated as:
$(y \times log(p) + (1 - y) \times log(1 - p))$

Assuming we have two classes C_1 and C_2, y[0,1] is the groundtruth and p the prediction for C_1 and 1-y and 1-p are the groundtruth and prediction for C_2.
The natural or base-e logarithm is used, meaning the unit for the cross entropy loss is nats. The standard implementation in keras was used.

Besides the loss, which is used by the model for training, several metrics are typically used to evaluate a binary classification model's performance. For all of those metrics we first need to compute the confusion matrix, which can be seen in Figure 25.

	Actually Positive	**Actually negative**	
Predicted Positive	True positive T_P	False positive F_P	Precision/Positive Predictive Value: T_P / T_P+F_P
Predicted Negative	False negative F_N	True negative T_N	Negative Predictive Value: T_N / T_N+F_N
	Sensitivity/Recall: T_P / T_P+F_N	Specificity: T_N / T_N+F_P	

Figure 25.: The confusion matrix.

Accuracy is usually used, as it is simple, easy to interpret and available in all frameworks by default. Accuracy can be useful in some cases, such as the predicted of unpaired and paired regions. For accuracy the keras implementation was used.

$$Accuracy = \frac{T_P+T_N}{T_P+T_N+F_P+F_N}$$

One case where accuracy is problematic is when classes are imbalanced. This is the case for predicting RNA pairing matrices, as most fields, often even over 90%, of the matrices are unpaired. So for example for a matrix which is about 99% unpaired, a model predicting totally unpaired matrices will have approximately 99% accuracy.
An alternative metric, which can be more useful in such cases, is the F1 score, the harmonic mean of the model's precision and recall.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} = \frac{T_P}{T_P + \frac{1}{2} \times (F_P + F_N)}$$

F1 has downsides in some scenarios, among other things it does not include the T_N, which is why the alternative Matthews Correlation Coefficient (MCC) can be a more reliable statistical estimate of performance [Chicco and Jurman, 2020].

$$MCC = \frac{T_P \times T_N - F_P \times F_N}{\sqrt{(T_P+F_P)(T_P+F_N)(T_N+F_P)(T_N+F_N)}}$$

It takes all cells of the confusion matrix into account. It is easily to interpret, as it lies between +1 and -1, with -1 representing a very poor performance, +1 representing perfect performance and 0 representing a performance equivalent to a random classifier.
F1 and MCC have been implemented as custom callbacks in keras.

For the sliding-window method and the LSTM the classification of T_P, T_N, T_P, F_N and F_P is based on the number of bases correctly predicted as paired or unpaired. Similar for all ResNet models the performances are based of correctly identified fields in the output matrix.
The output values for all methods are between 0 and 1 with all values below 0.5 counting as unpaired and all above or equal to 0.5 as paired.

As several aspects of RNA folding are known one aspect of performance evaluation is based on the models ability to learn these properties.

One easily testable property is how the number of base pairs is correlated with the sequence length, here predictions by the deep learning models are compared with the number of base pairs predicted by RNAfold.
Again a threshold of 0.5 is used to determined which bases are paired.

Another basic property of RNA folding matrices is that they should be symmetric, so if base i is predicted to pair with base j, base j should also be predicted to pair with base i.

This is pretty intuitive, however it is not guaranteed with this implementation of structure prediction.
In SPOT-RNA this is solved by only using the upper triangular matrix for final predictions. However, this introduces a decision about base pairs, which is done after the model training. If results are too different between the two triangular matrices, this could have a considerable impact on the final predicted structure.
As most of the matrix will consist of zeros, meaning the bases are unpaired, the symmetry of these positions will be always very high, so only the symmetry of predicted pairs will be analyzed.

Another constraint for RNA secondary structures is that one base can be paired with maximally one other base, ignoring triple base pairs. Triple base pairs are included in SPOT-RNA, however this can be problematic as datasamples for such pairings are scarce. The total percentage of bases which are correctly only paired maximally once is computed. as well how often bases are paired, if they form more than one base pair.
The presented results are based on the triangular matrix after applying the threshold of 0.5.

In RNA secondary structures base pair types are usually restricted to only canonical base pairs, which are pairings between Adenine and Uracil, Cytosine and Guanine as well as Guanine and Uracil.
The pairing types are evaluated based on the triangular matrix after applying the threshold of 0.5.

The last constraint is that the distance between two paired bases has to be at least four bases, so that a minimum loop size of 3 is guaranteed. This is also evaluated based on the triangular matrix after applying the threshold of 0.5.

The topology of the predicted RNA structures was compared as well, meaning the percentages of bases in different loop types and the average length of loops and stems.
The lengths of interior loops and multi loops are based on the individual parts of the loops.
The triangular matrix after applying the threshold of 0.5 was used as basis, with two additional steps to ensure no base pairs between multiple bases exist. The first step was to use only the maximum scoring base pair for each row. The second step was to only include base pairs for two bases, which are no yet paired. So if the base at position 10 and the base a position 20 are predicted as pairing and the base at position 15 is predicted as pairing with the base at position 20 as well, the base pair between 15 and 20 is not included.
As this is by no means an optimal solution, the topology evaluation was only done for RNA sequences with lengths from 30 to 100, where pairings between multiple bases are predicted very rarely.
After the removal of multiple base pairings, pseudoknots are removed as well and the respective structures are determined, both steps are done using the Python interface of the ViennaRNA package [Lorenz et al., 2011].

Part III.

Results and Discussion

9. Predicting paired/unpairedness

9.1. Performances

The first models were based on the idea of predicting paired or unpaired regions of the RNA.
A sliding-window approach is used as a baseline. Two different network types are used, one simple feed forward and one convolution network, and window-sizes 15, 35 and 71.
Three different BLSTM networks are tested, one with 1 layer and 40 neurons, another with 1 layer and 80 neurons and a third with 3 layers and 40 neurons.

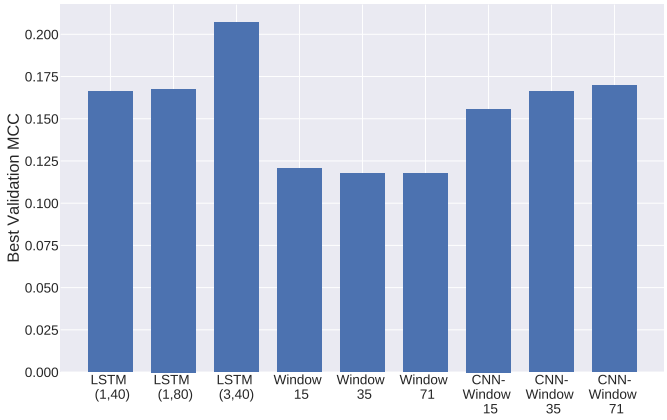

Figure 26.: The maximum validation MCC of all models trained on 80000 sequences of length 70 for 100 epochs for a validation set of 2000 sequences of length 70. Starting from left the first three networks are the LSTM networks, the next three are the simple feed forward sliding-window networks and the last three are the convolutional sliding-window networks.

Modeltype	Parameters	Epochs	Accuracy	F1	Loss	Mcc
LSTM	1 Layer, 40 Neurons	43	0.667	0.594	0.609	0.166
	1 Layer, 80 Neurons	27	0.664	0.589	0.612	0.168
	3 Layers, 40 Neurons	38	0.676	0.609	0.604	0.207
Sliding Window	Window 15	89	0.654	0.559	0.623	0.120
	Window 35	94	0.659	0.559	0.620	0.118
	Window 71	59	0.661	0.569	0.618	0.118
CNN Sliding Window	Window 15	67	0.660	0.588	0.616	0.156
	Window 35	65	0.666	0.586	0.609	0.166
	Window 71	30	0.668	0.580	0.608	0.170

Table 2.: The performances on the validation set of 2000 sequences of length 70 for all models trained on 80000 sequences of length 70 for 100 epochs.
After 100 epochs the best performing model is chosen based on maximum validation MCC.
The epoch in which this performance is reached can also be seen in the table.
The metrics used are accuracy, F1, loss and MCC. All values are rounded to three decimal places.

As can be seen in Figure 26 and Table 2, all models perform quite poorly, with MCC values between 0.118 and 0.207 and accuracy values between 66% and 67,6%
In general the sliding-window approach with a simple feed forward network performs the worst, with a very small increase in performance for larger window sizes.
The sliding-window approach with a convolutional network performs better and for the two larger window sizes it actually performs similar to the two single layer LSTM, with window size 71 it even outperforms them slightly.
The best performing model is the LSTM with 3 layers, indicating that the performance could actually be further increased with more depth. Tests with more complex, deeper models were planned for this thesis, however as discussed in the methods these models suffered from instabilities.
One interesting aspect is also the training time (in epochs) needed to reach the best performance.
A general trend which is visible is that better performing models also train faster, reaching their maximum performance earlier. For the sliding-window approaches the models with the largest window size also reached maximum performance earlier.

9.2. Learning curves

The next step was to figure out, if the poor performance could be enhanced by using more data.
This is generally done by using learning curves, where you train the same model with increasingly large subsets of the original training set, while evaluating them with the original validation set.
As can be seen in Figure 28 and Figure 29, nearly all sliding windows reach their maximum performance already with quite small training sets of 10000 sequences. One exception is the sliding-window model using a simple feed-forward network with window size 71, where the validation performance still increases slightly until 50000 sequences are used for training.
As can be seen in Figure 27 the LSTM models show more difference in learning curves. The LSTM model with 1 layer and 40 neurons behaves like most of the sliding-window models, reaching maximum performance already with quite small training sets of 10000 sequences.
This indicates that training sets of around 10000 sequences or even a bit less are enough to reach maximum performance, which is also the reason why the next tests were performed with less data.

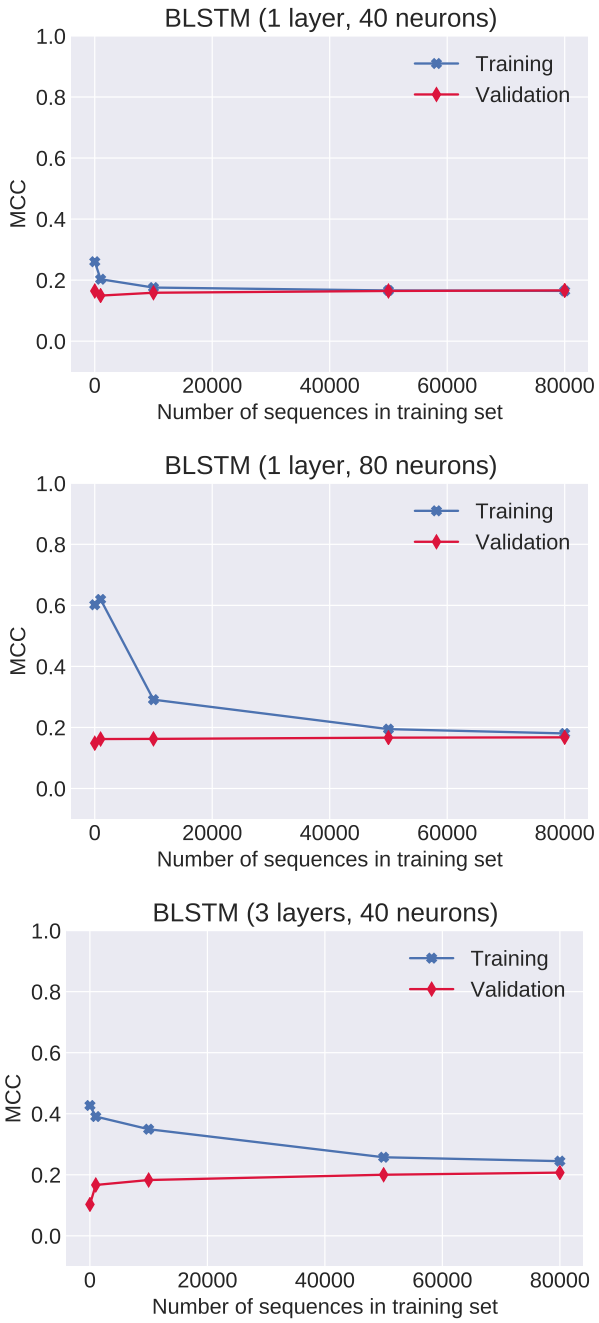

Figure 27.: Learning-curves for all LSTM models, based on training on subset the training set with 80000 sequences all validated on the 20000 sequences validation set. The subsets used were 10, 100, 1000, 10000, 25000 and 50000 sequences.

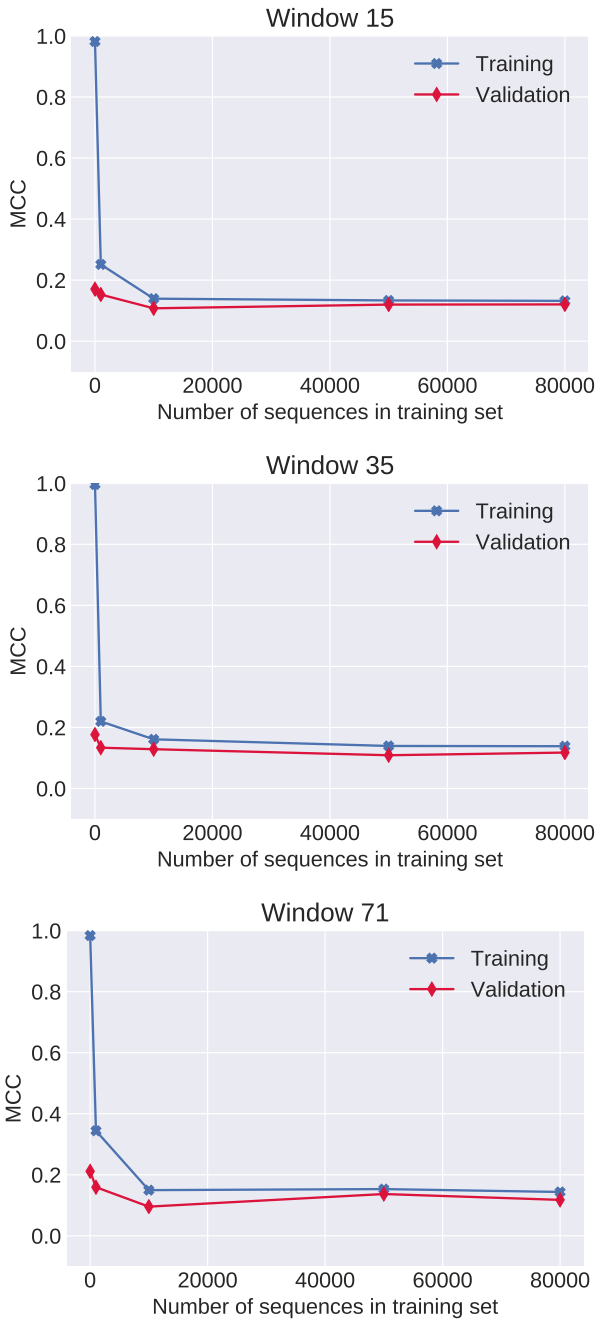

Figure 28.: Learning-curves for all sliding-window models using simple feed-forward networks, based on training on subset the training set with 80000 sequences all validated on the 20000 sequences validation set. The subsets used were 10, 100, 1000, 10000, 25000 and 50000 sequences.

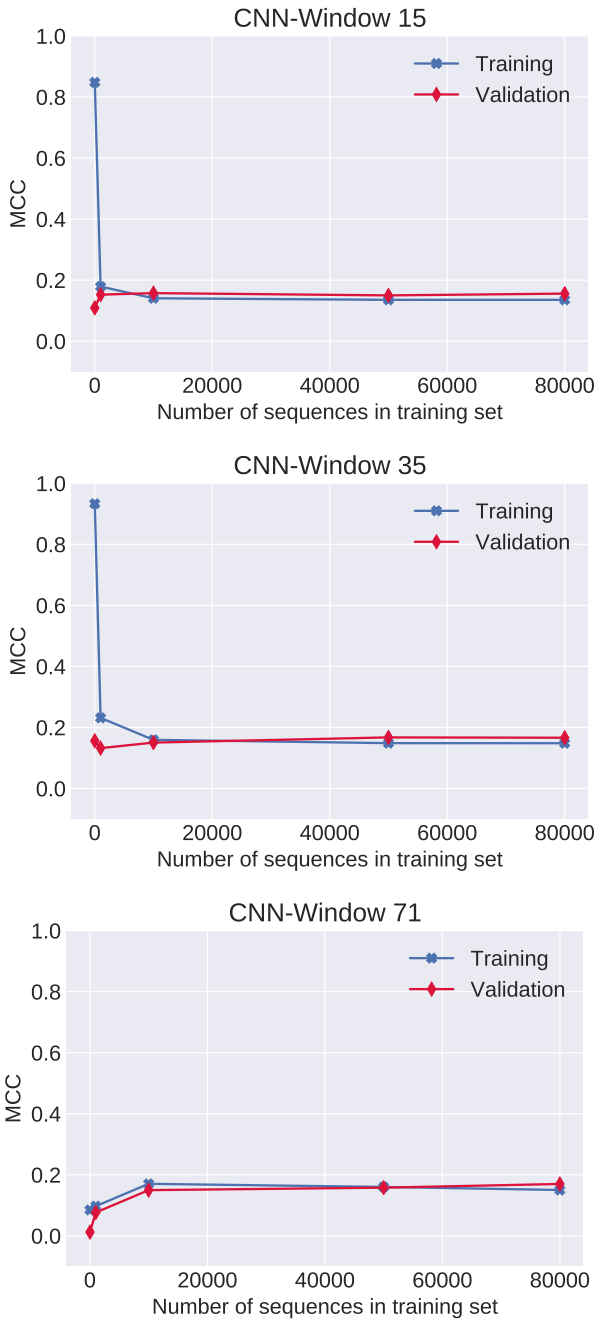

Figure 29.: Learning-curves for all sliding-window models using convolutional networks, based on training on subset the training set with 80000 sequences all validated on the 20000 sequences validation set. The subsets used were 10, 100, 1000, 10000, 25000 and 50000 sequences.

9.3. Variability of results

In machine learning the performance can greatly vary between datasets and also between different training runs for the same dataset, due to the random factors in the training process, such as initialization.
As can be seen in Figure 30 the variation when using different datasets is rather small, especially for LSTM models. The overall performance differences between the different models also stay pretty similar to the first test with the LSTM with 3 layers and 40 neurons outperforming all other models. The only major difference is that the convolutional sliding-window network with window size 71 performs slightly worse than the other window-sizes.

The results for the different runs are very similar, as can be see in Figure 31.
The LSTM model with 3 layers and 80 again outperforms all other models and the LSTM models generally show less variability than the sliding-window models.

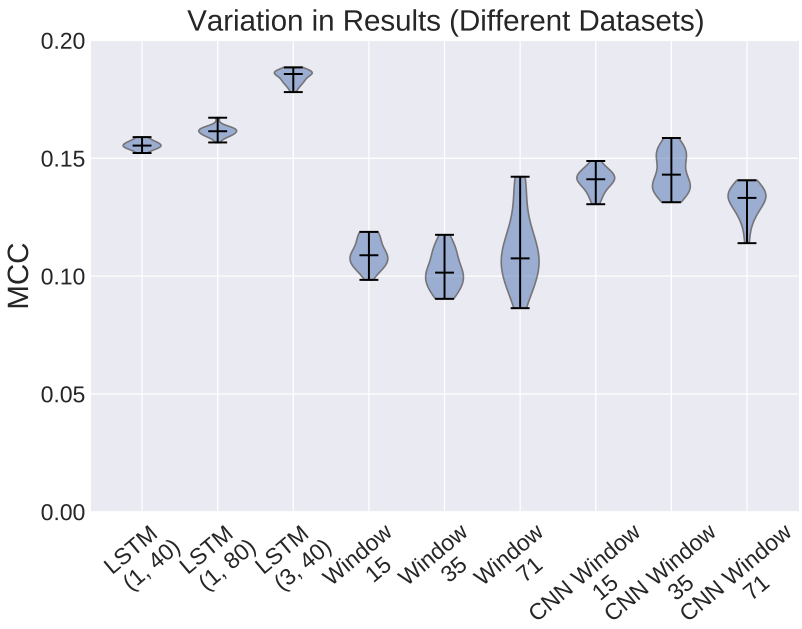

Figure 30.: A violin plot of all models trained on 20 different datasets with 8000 sequences of length 70 for 100 epochs for a validation set of 2000 sequences of length 70 each.
Starting from left the first three networks are the LSTM networks, the next three are the simple feed forward sliding-window networks and the last three are the convolutional sliding-window networks.

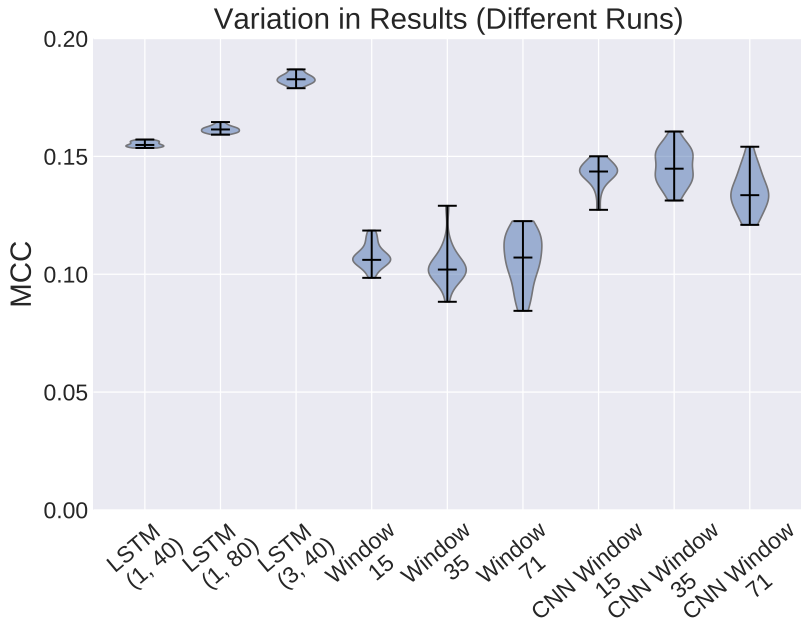

Figure 31.: A violin plot of all models trained 20 times on the same dataset with 8000 sequences of length 70 for 100 epochs for a validation set of 2000 sequences of length 70.
Starting from left the first three networks are the LSTM networks, the next three are the simple feed forward sliding-window networks and the last three are the convolutional sliding-window networks.

9.4. Recap

From all testsit can be seen, that both the baseline sliding-window models and the LSTM models perform very poorly. And while one of the LSTM models outperforms all of the sliding-window models, the increase in performance is minor.
The results also indicate that the performance can not be improved notably by more data or more capacity. The results were also consistent for different datasets and different training runs.
This leads to the conclusion that the paired/unpaired classification seems to be a non suitable simplification of secondary structure, different to for example the way secondary structure is a simplification for the full tertiary structure.

10. Predicting base-pairing matrices

10.1. Results and learning curves

As the simplification of secondary structure to paired/unpaired has been unsuccessful, another representation of secondary structure was needed. The architecture which was chosen for implementation, is an approach combining a ResNet with a 2D-BLSTM, based on the already published SPOT-RNA [Singh et al., 2019].
The output is a matrix predicting for each position (i,j), if the bases i and j are paired. The input is a matrix of concatenated one-hot encodings.
The model numbering is based on the original publication, where five models with different architectures have been used. The basic distinction of the models is that model 0 and 1 are have a ResNets architecture combined with Fully Connected layers. Model 1 has less ResNet blocks and ResNet filters. Model 3 is a ResNet architecture combined with a 2D-BLSTM.

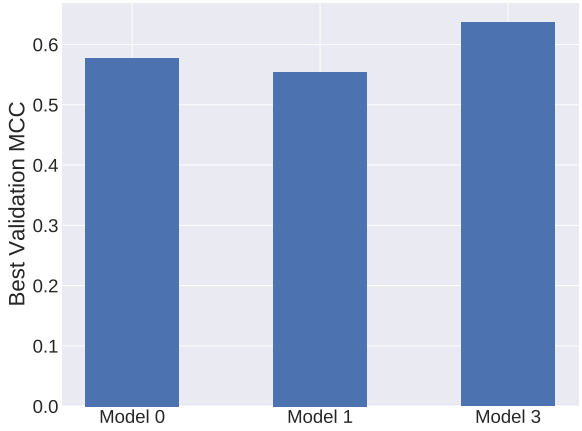

Figure 32.: The maximum validation MCC of all models trained on 8000 sequences of length 70 for 50 epochs for a validation set of 2000 sequences of length 70.

From Figure 32 it is visible that these models perform much better, with MCC values of 0.554 for model 0, 0.679 for model 1 and 0.653 for model 3. This is also very similar, though a bit lower, than the reported MCC values of 0.681 for model 0, 0.679 for model 1 and 0.653 for model 3 or SPOT-RNA after initial training.

The difference in performance is interesting, as for our dataset model 0 performs the worst, model 1 is in the middle and model 3 performs the best, while the order in the original publication is actually reversed.

Again learning-curves were done for all models, see Figure 33.
And while the overfitting in the training set is larger than for the previous models, the validation performances reach their peak before the full set of sequences is used. It can also be seen that the difference in training and validation performance is consistently higher for better performing models, where the better performing model does not necessary need to be more complex, see model 0 and 1.

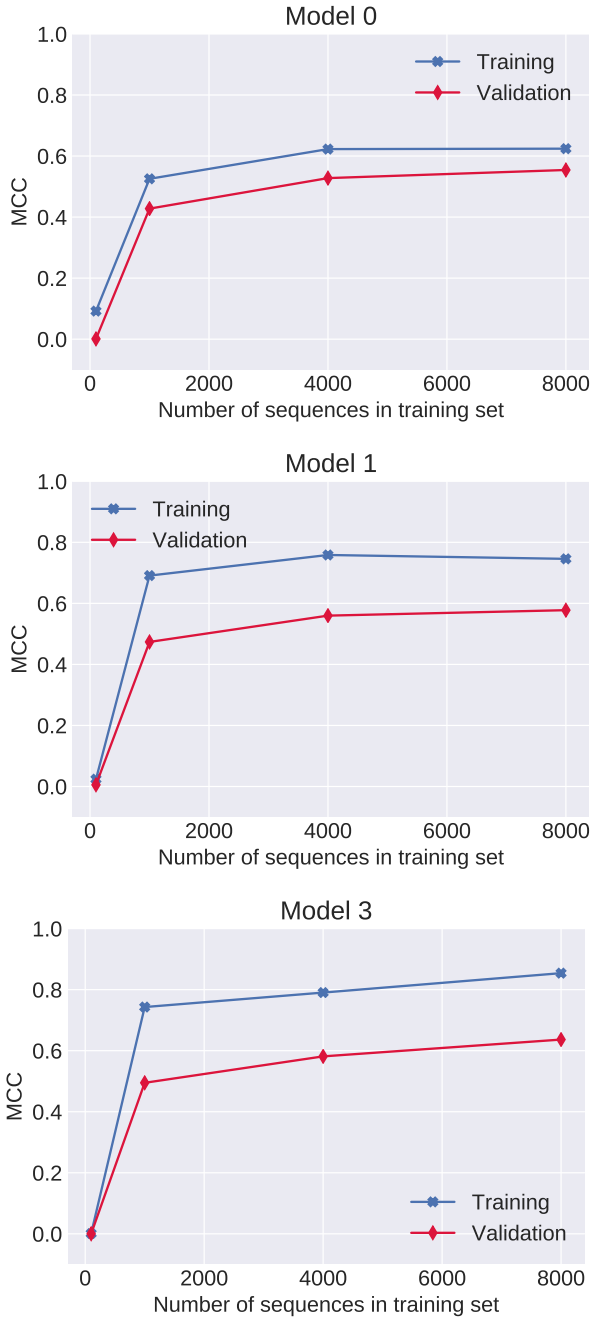

Figure 33.: Learning-curves for all models using simple feed-forward networks, based on training on subsets of the training set with 8000 sequences all validated on the 2000 sequences validation set. The subsets used were 100,1000 and 4000 sequences.

10.2. Dataset tests

As has been discussed, dataset variability plays a large role in machine learning model performance.
The usage of different datasets also makes the different models and approaches difficult to compare, especially as no consistent criteria for validation and test sets are used.
These concerns are not only relevant for comparing different machine learning approaches, but also for users, who want to use the models to predict RNA structures.

One aspect which can be easily tested within the settings of this thesis is the sequence length.
Sequence length is also relevant for real world RNA datasets, as they usually have peaks at specific lengths with very few sequences of other lengths, as can be seen in Figure 34.
So far all models in this thesis have been trained on sequences with length 70, for this test model 1 has been trained on four differently distributed datasets with sequence length between 25 and 100, see Figure 35.
Model 1 has been chosen as it was the better performing and less complex model of the two ResNet + FC networks. And while it would be interesting to compare model 3, the ResNet + 2D-BLSTM network, with model 1, unfortunately GPU memory restrictions have not allowed for this.

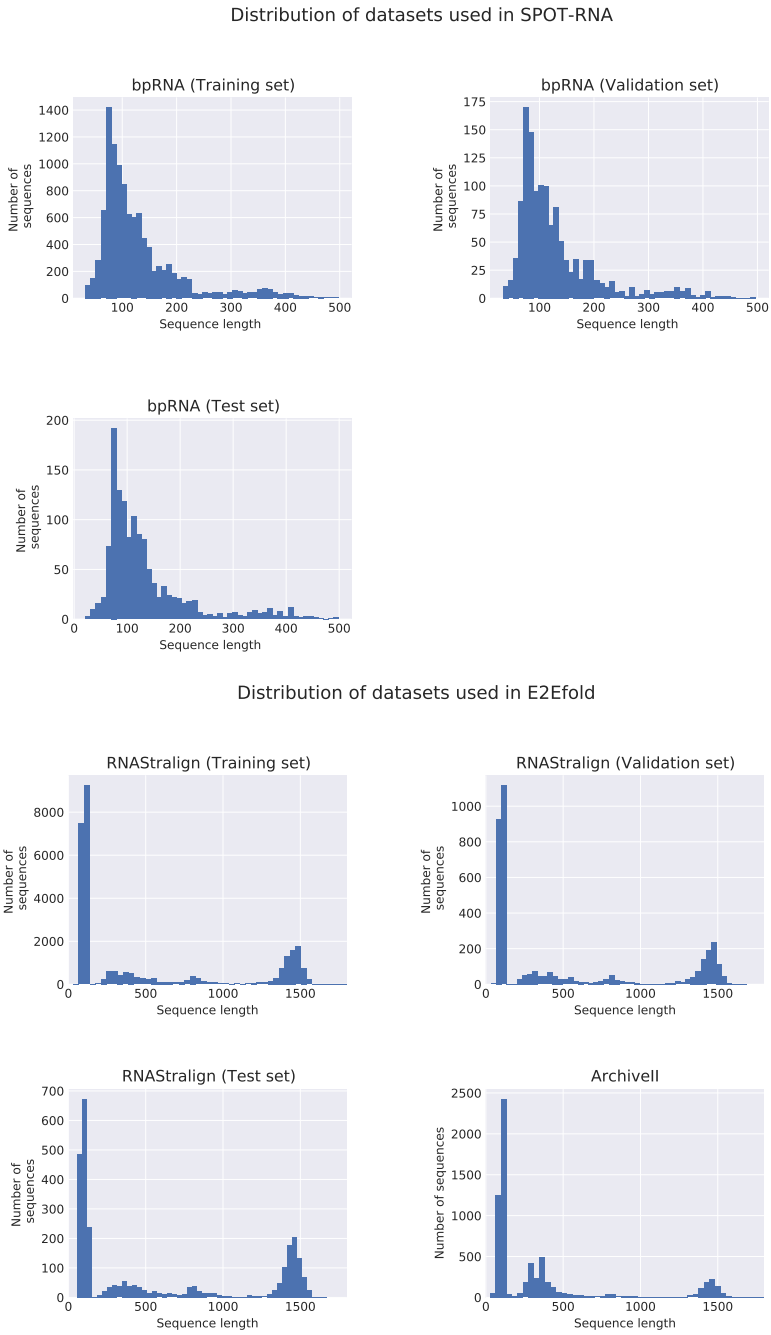

Figure 34.: Different datasets used in recent publications for machine learning for RNA secondary structure prediction. The first three plots are from SPOT-RNA [Singh et al., 2019] based on the bpRNA dataset [Danaee et al., 2018]. The next four plot are from E2Efold [Chen et al., 2020b], the first three plots are based on the dataset RNAStralign [Tan et al., 2017] and the fourth plot is based on the dataset ArchiveII [Sloma and Mathews, 2016], which is used as additional test set.

Figure 35.: As a reminder, these are the four different distributions used in this thesis. The plots are based on the training sets, however the validations sets have the same overall distributions.

Training dataset	Validation dataset 1	2	3	4	Performance on training set
1	**0.64**	0.59	0.61	0.71	0.72
2	0.61	**0.58**	0.59	0.68	0.66
3	0.64	0.60	**0.62**	0.70	0.71
4	0.63	0.57	0.59	**0.75**	0.87

Table 3.: The performances of all combinations of training and validation datasets for the four distributions shown in Figure 35.
The diagonal in red shows the performance, when training and validation dataset have the same distribution.

In the diagonal in Table 3 it can be seen that the reported performances for all these models on their respective validation set are notably different from a minimum of 0.58 to a maximum of 0.75
From these results one might chose the model trained on dataset 4, however this high performance originates from the general trend that structures are easier to predict for shorter sequences.
For all other datasets the model trained on dataset 3 performs best or equally to the corresponding model.

Although the datasets were restricted to a rather small range of lengths, from 25 to 100 bases, a notable differences is already observable. In this case the performance differences stem mainly from the fact that short sequences are easier to predict.
This indicates that when using machine learning models, RNAs whose lengths differ from the ones making up most of the original training set might not be accurately predicted. This will be especially notable, if the sequences to be predicted are longer than ones making up most of the original dataset.

To explore this further validation sets with sequence lengths from 30 to 250 were used to observe how the usage of different models and training sets influences performance, even for sequence lengths which were originally not in the training set.
In Figure 36 the performances for each model and dataset are plotted.
The models only trained on sequences with length 70 show a peak in performance at this length.
Both model 0 and 1 (the two ResNet+FC models) also showed good performances for shorter sequences, while model 3 (the ResNet+2D-BLSTM model) showed a strong drop in performance for both shorter and longer sequences.
All models trained on the differently distributed datasets with length 25-100 perform relatively similar, with stronger performances for shorter sequences and lower performance for longer sequences. The main drop in performance can already be seen for longer sequences, which are still in the training set. For example for dataset 1 the MCC drops from 0.76 for sequence length 30 to 0.53 for sequence length 100 and to 0.39 for sequence length 250.
A small effect which different datasets have in generalization can be seen when comparing dataset 3 and dataset 4.
While the model trained on dataset 3 (consisting of longer sequences) shows a lower performance for shorter sequences than the model trained on dataset 4 (consisting of shorter sequences), it performs better on the longest sequences.
For sequences over 100 bases the model trained on dataset 3 performs the best out of all models and datasets, while the model trained on dataset 4 performs worst.

This example indicates, that the effect different dataset distributions have on the generalization ability can be further amplified by the specific model architecture.
This should also be kept in mind when making architecture decisions, as the best performing model might not always be the optimal choice due to poor generalization ability.
When validation and also test datasets have the same or similar distributions this can remain undetected during model evaluation.
This further emphasises the need for consistency in evaluation of different machine learning approaches as well as clear communication of chosen datasets and criteria for divisions into training, validation and test set.

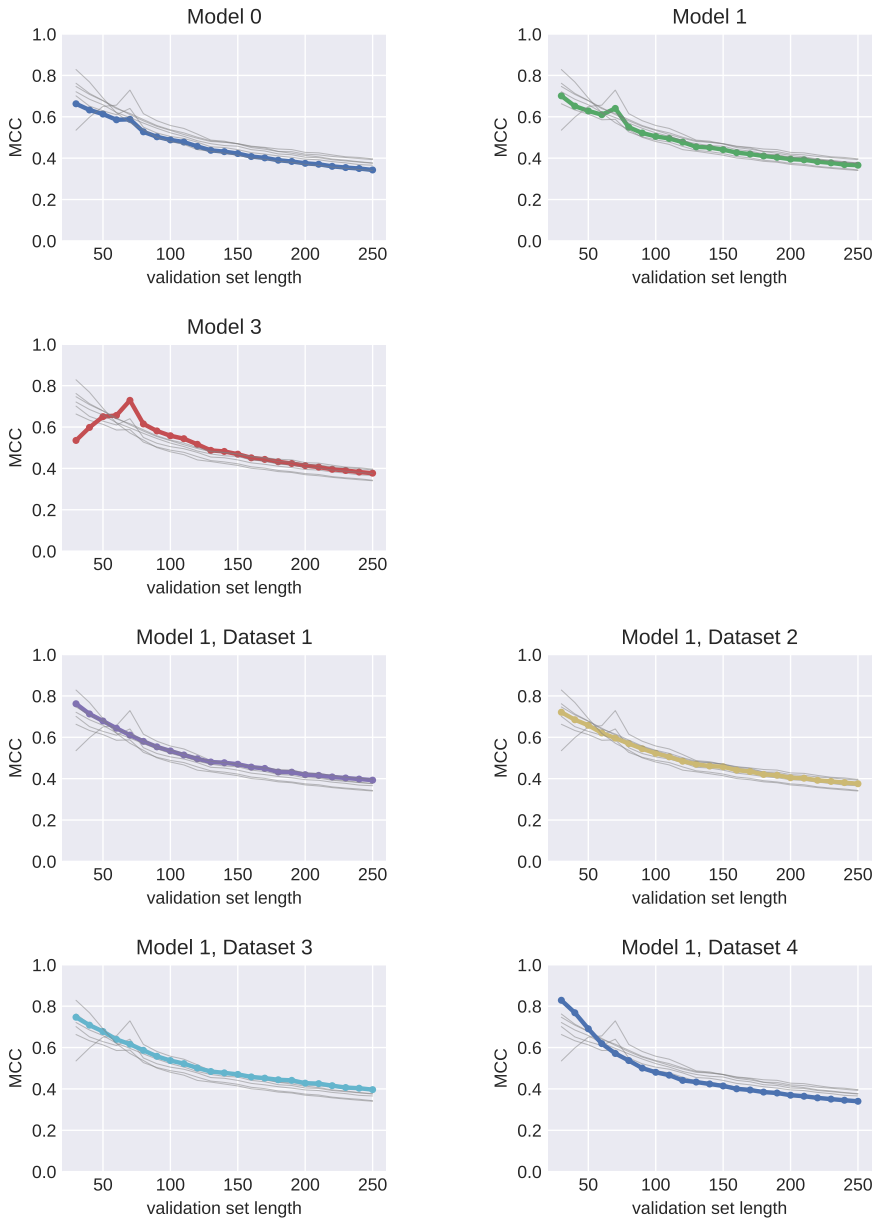

Figure 36.: MCC values for test sets with sequences of length 30 to 250 consisting of 2000 sequences each.
The first three plots are for model 0, 1 and 3 trained on the set of 8000 sequences of length 70. The other four plots are for model 1 trained on dataset 1-4, with different distribution of sequences with length 25-100 and a training set size of 10000 sequences.

10.3. Folding properties

So far the performance has always been measured elementwise on the output matrices. However, several aspects of RNA folding are already well known, so machine learning models can be evaluated based on their ability to learn or represent these constraints correctly.
Generally, there are three ways to ensure that the constraints are fulfilled, which are post-processing, learning from data and iterative optimization.
In this case the approach of learning them from data will be discussed.

The constraints can be split into two categories based on the context needed to predict base pairs fulfilling the constraints.
The context is either global, meaning the model needs overview over the whole sequence or local, meaning the model can make predictions fulfilling these constraints separately for each base pair.

10.3.1. Global features

One global property is how the number of base pairs is correlated with the sequence length. The number of base pairs increases with sequence length, with a high linear correlation coefficient of 0.86 or one base pair for every four nucleotides [Zhao et al., 2018].
To correctly represent this the prediction method needs to have a global overview over predictions, otherwise a quadratic output would be expected when predicting secondary structures in matrix representations.
As models were trained with structures predicted by RNAfold, the number of base pairs predicted by RNAfold was used as ground truth.
Results are based on a threshold of 0.5, which was used for training, and only using the upper triangular matrix to decide which bases are paired, as this was the approach used in SPOT-RNA.
In Figure 37 the results and the corresponding functions are displayed.
All models predict too many base pairs for longer sequences, with the models only trained on sequences of length 70 deviating earlier from the ground truth. For these three models model 3, the ResNet+2D-BLSTM model, performs best although it slightly underpredicts the number of base pairs for shorter sequences. Model 1 performs the worst and even predicts more base pairs than bases for very large sequences.
The models trained on a datasets 1-3 accurately predict the number of base pairs in the range of sequence lengths they were trained on. The model trained on dataset 4, which consisted mainly of shorter sequences, deviates only slightly before reaching a sequence length of 100.

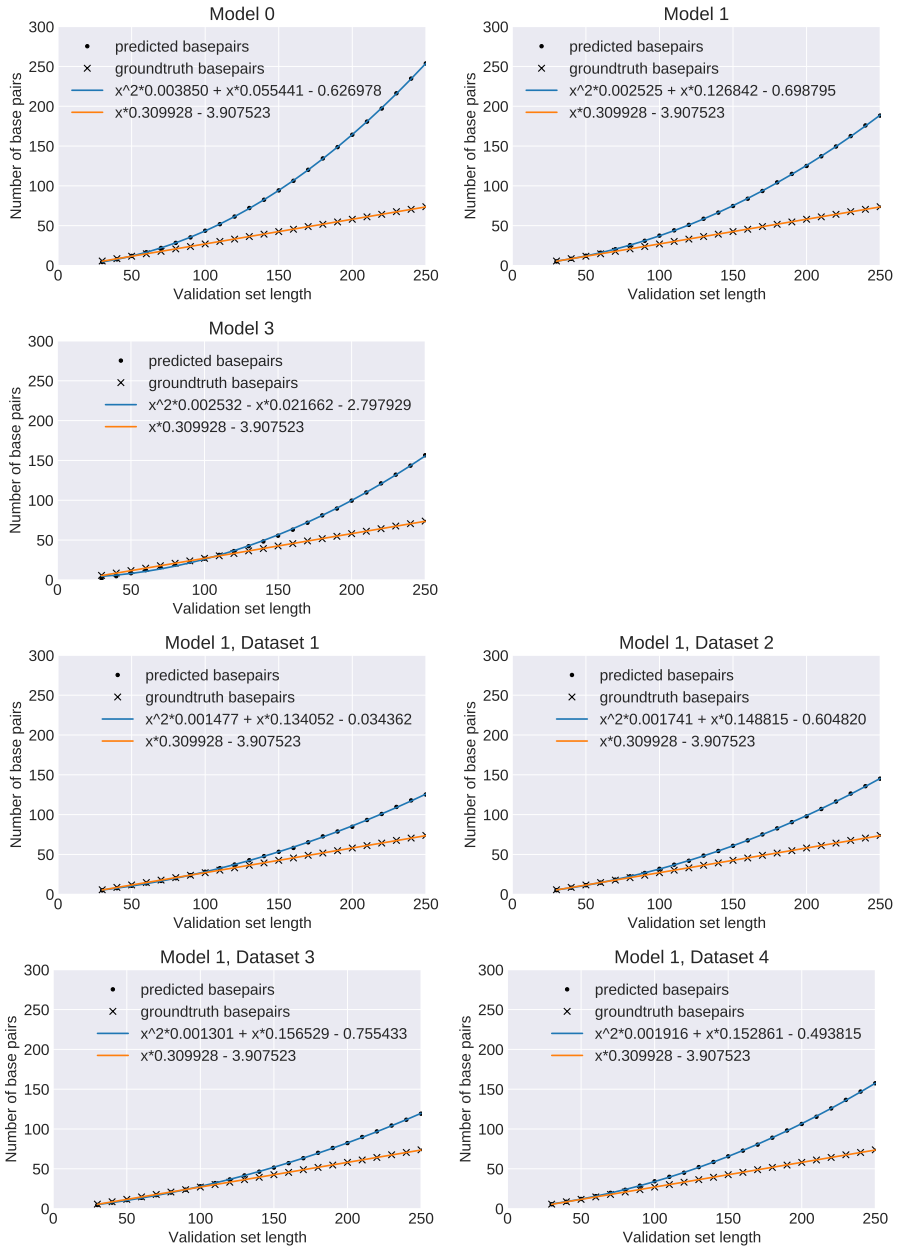

Figure 37.: The average number of base pairs per sequence for test sets with sequences of length 30 to 250 consisting of 2000 sequences each.
The ground truth, which is the MFE structure predicted by RNAfold, can be fitted by a linear function while the predictions are fitted by a quadratic function.
A threshold of 0.5 is used to decide which positions are paired or unpaired.
The first three plots are for model 0, 1 and 3 trained on the set of 8000 sequences of length 70. The other four plots are for model 1 trained on dataset 1-4, with different distribution of sequences with length 25-100 and a training set size of 10000 sequences.

Another global property of RNA folding matrices is, that they should be symmetric, so if base i is predicted to pair with j base j should also be predicted to pair with base i.
As mentioned before this is solved in SPOT-RNA by only using the upper triangular matrix for final predictions. However, this introduces a decision about base pairs which is done after the model training, and if results are too different between the two triangular matrices this could have a considerable impact on the final predicted structure.
As most of the matrix will consist of zeros, meaning the bases are unpaired the symmetry of these positions will be always very high so only the symmetry of predicted pairs were analyzed.
In Figure 38 the average percentage of symmetric base pairs is displayed. Although the numbers are mainly in the 90% range, the base pairings are never fully symmetrical. Additionally a strong overfitting to sequences of length 70 for model 3 can be seen. A good performance for very short sequences is only seen for model 1 trained on dataset 4, which is expected as dataset 4 mainly consists of short sequences

The last property of RNA secondary structures is that one base can be paired with maximally one other base, especially when only considering canonical base pairs. This means that the row and column sums of the matrix have to be between 0 and 1.
The average number of row sums ≤ 1 based on the upper triangular matrix after applying the threshold of 0.5 are displayed in Figure 39.
The result is consistent with the result for the number of base pairings in regard to sequence length.
While all models perform worse on longer sequences model 3 and all models trained on dataset 1-4 perform considerably better than model 1 and especially model 0. The models trained on dataset 1 and 3 and model 3 perform best reaching close to 100% row sums ≤ 1.
In Figure 40 the number of base pairs, which are formed in case a base pair forms more than one are depicted.
In the original SPOT-RNA paper multiple base pairings for one base are not accounted for, as the authors aim to include triple base pairs in their predictions.
As already mentioned, this is problematic, as the data samples from which these base pairs can be learned are sparse.
This is especially problematic, as the results in this thesis show, that the models are not able to learn the correct number of base pairing per base from the training set, although the same property can be seen throughout all data samples. They would thus probably not be able to learn to correctly identify the correct instances to include triple base pairs.
Additionally triple base pairs are not usually two different Watson-Crick pairs, but for example one Watson-Crick pair and one Hoogsteen pair [Gautheret et al., 1995].

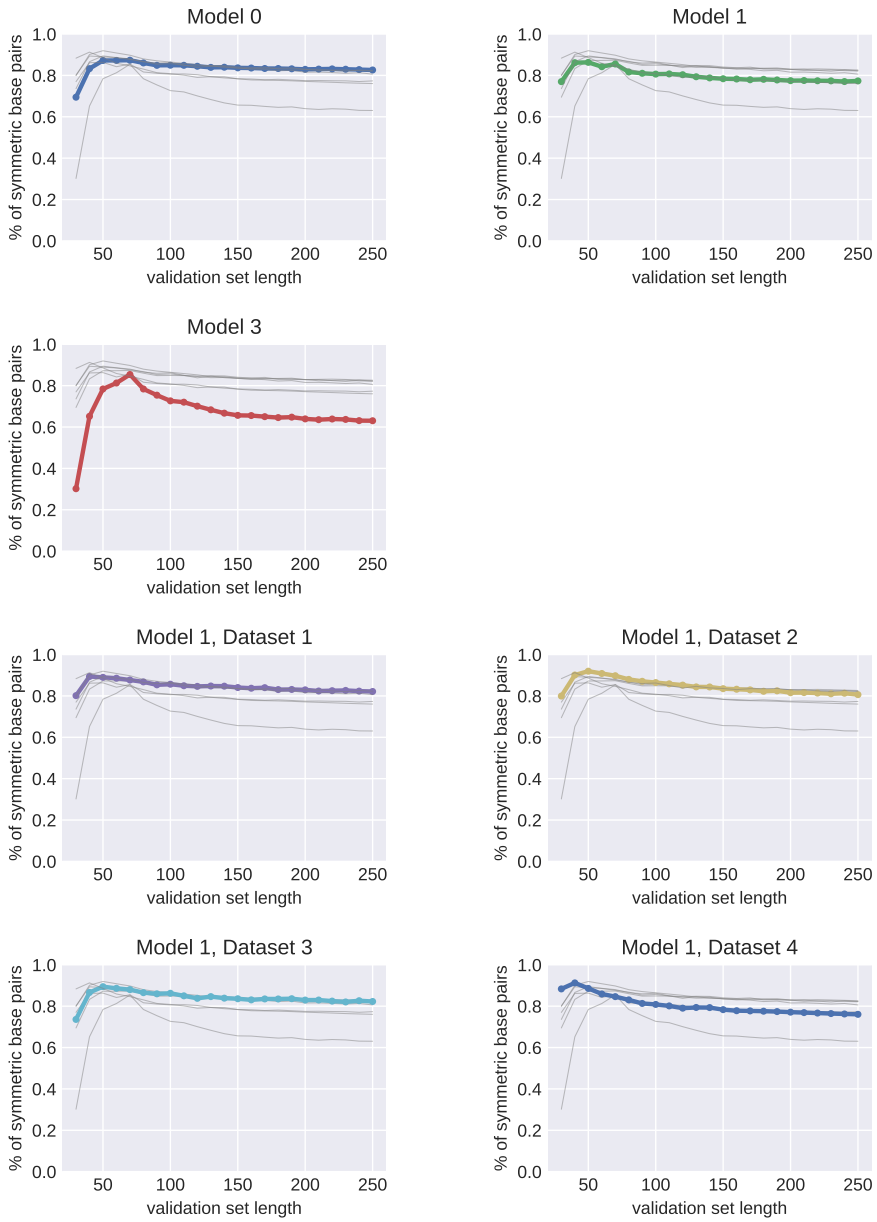

Figure 38.: The average percentage of correctly symmetrical base pair predictions for test sets with sequences of length 30 to 250 consisting of 2000 sequences each. The first three plots are for model 0, 1 and 3 trained on the set of 8000 sequences of length 70. The other four plots are for model 1 trained on dataset 1-4, with different distribution of sequences with length 25-100 and a training set size of 10000 sequences.

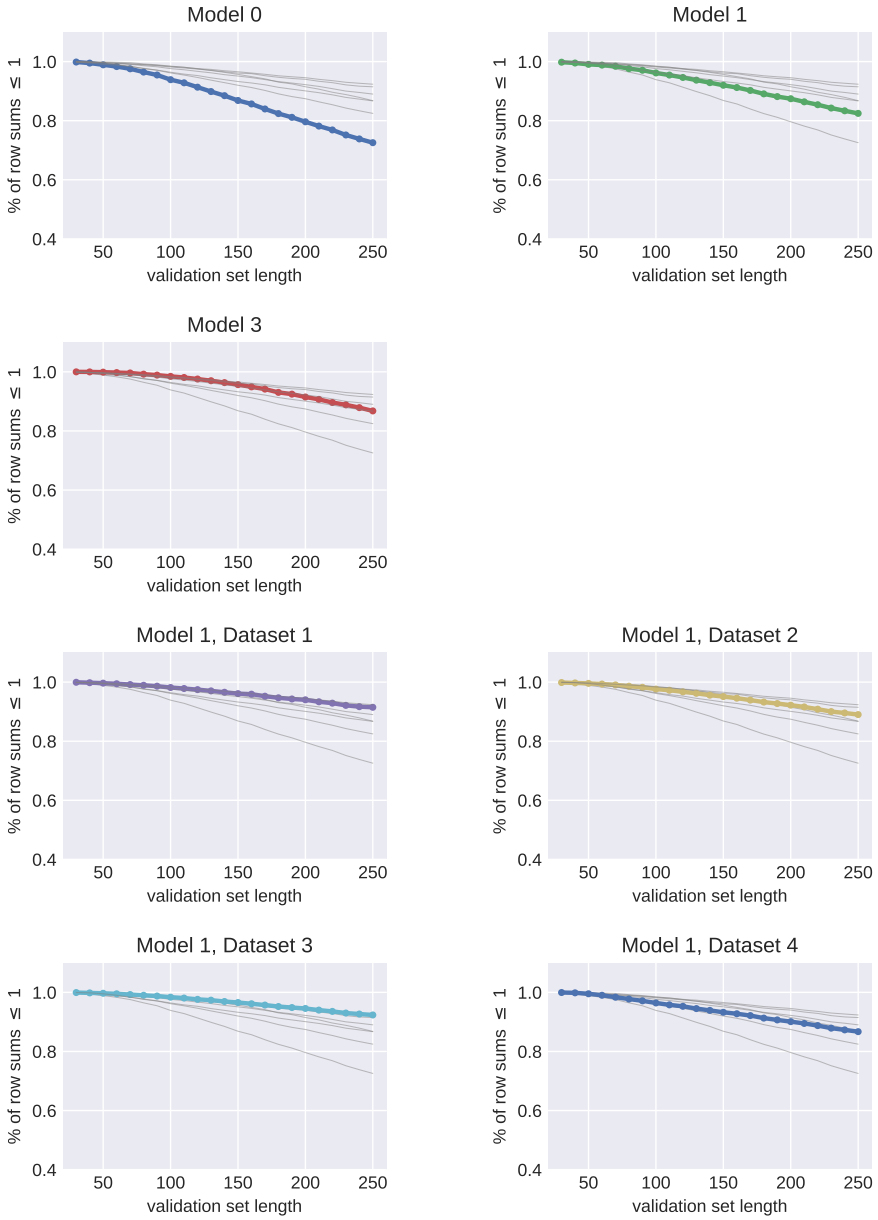

Figure 39.: The percentage of rows with sums ≤ 1 for test sets with sequences of length 30 to 250 consisting of 2000 sequences each.
Results are based on rounded upper triangular matrices. Rounded meaning that the threshold was already applied, so that positions with values < 0.5 are predicted as unpaired (=0) and positions ≥ 0.5 are predicted as paired.
The first three plots are for model 0, 1 and 3 trained on the set of 8000 sequences of length 70. The other four plots are for model 1 trained on dataset 1-4, with different distribution of sequences with length 25-100 and a training set size of 10000 sequences.

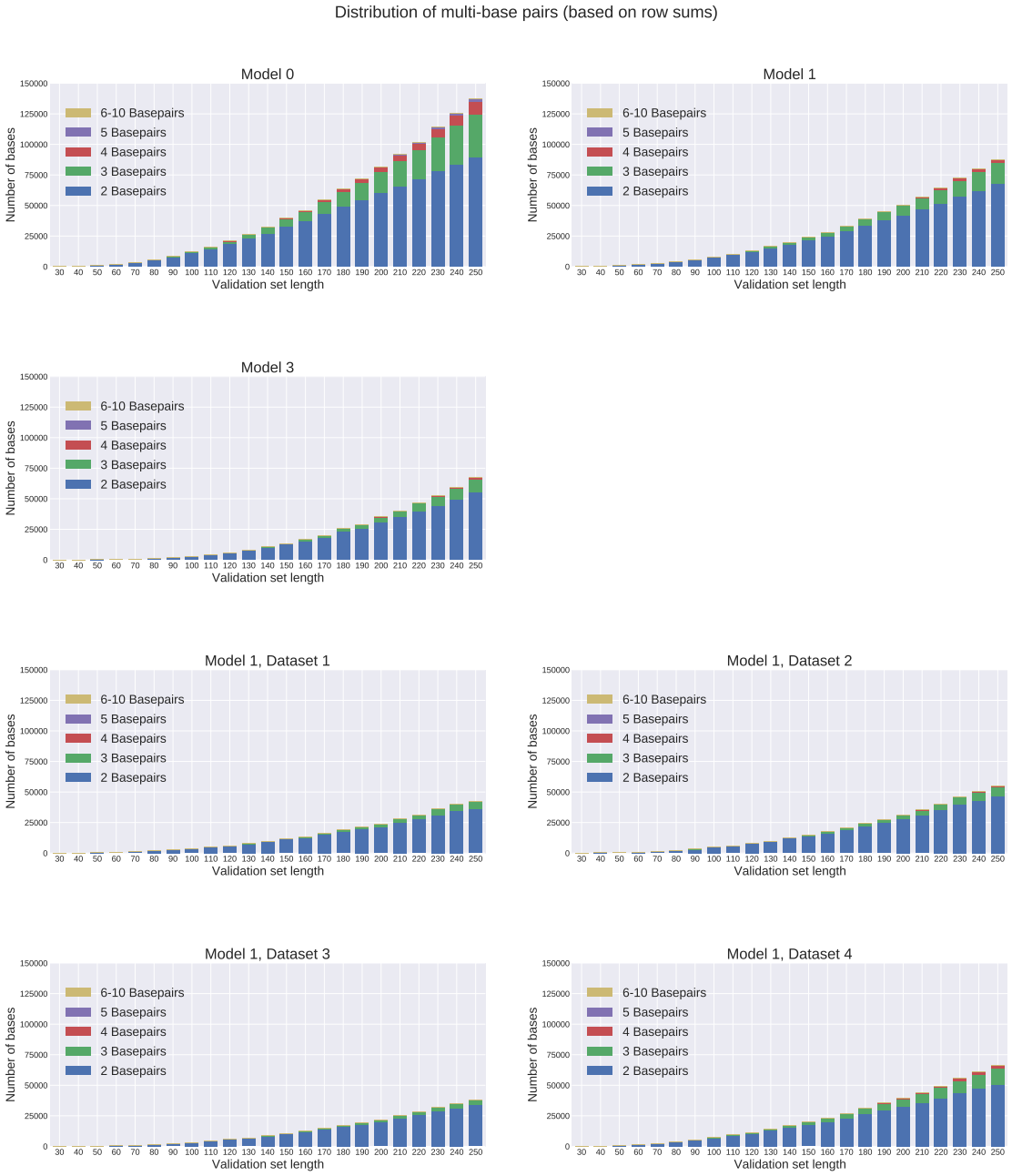

Figure 40.: A breakdown of all rows with sums > 1 for test sets with sequences of length 30 to 250 consisting of 2000 sequences each.
Results are based on rounded upper triangular matrices. Rounded meaning that the threshold was already applied, so that positions with values < 0.5 are predicted as unpaired and positions ≥ 0.5 are predicted as paired.
The first three plots are for model 0, 1 and 3 trained on the set of 8000 sequences of length 70. The other four plots are for model 1 trained on dataset 1-4, with different distribution of sequences with length 25-100 and a training set size of 30000 sequences.

10.3.2. Local features

Usually RNA secondary structure prediction is restricted to canonical base pairs (C,G), (G,C), (A,U), (U,A), (G,U) and (U,G).
While the types of base pairs are not restricted in the machine learning approach presented in this thesis, the input, which is based on predictions from RNAfold, only contains these types of base pairs.
As can be seen is Table 4 this constraint is represented very well .

Modeltype	Training set	Base Pairs (Total)	False Base Pair - Combinations (Total)	False Base Pair - Combinations (%)
Model 0	Length 70	4578103	272	0.00005941
Model 1	Length 70	3572823	672	0.00018809
	Dataset 1	2491972	55	0.00002207
	Dataset 2	2852354	74	0.00002594
	Dataset 3	2409373	45	0.00001868
	Dataset 4	3077325	23	0.00000747
Model 3	Length 70	2822106	291	0.00010310

Table 4.: The statistics for all models for test sets with sequences of length 30 to 250 consisting of 2000 sequences each.
Results are based on rounded upper triangular matrices. Rounded meaning that the threshold was already applied, so that positions with values < 0.5 are predicted as unpaired and positions ≤ 0.5 are predicted as paired.
"False Base Pair-Combinations" includes all predicted base pairs which are non-canonical base pairs, so other pairings than A-U, C-G or G-U.
The percentages are rounded to eight decimal places.

The second local constraint is the minimum loop size of three bases, meaning two bases have to have a distance of at least four, if they are paired.
For all 22 sets with sequence lengths from 30 - 250 with 2000 sequences each and all models, only 2 base pairs do not fulfill this constraint.
In this case the model is also able to learn constraints present in the data.

In general, it can be seen that local constraints are much easier for the machine learning models to learn and to correctly represent in predictions.

10.3.3. RNA topology

As was described in the introduction, RNA secondary structure can be used as a basis to predict tertiary structure. This can be limited by the ability of secondary structure prediction methods to predict an overall correct topology, meaning the formation of structural elements such as loops, stems and bulges. This is still a challenge as the correct prediction of topologies is not strongly correlated with consistency in base pairs and only a few wrongly predicted base pairings can change the overall topology notably [Zhao et al., 2018]
In the following the overall topologies of the predicted RNA structures will be compared, meaning the percentages of bases in different loop types and the average length of loops and stems. The results are based on trends in the overall datasets not on individual sequences.

In the first part the percentages of bases, which are in the different structural elements are analyzed and compared to the ground truth, which are MFE structure predictions from RNAfold, see Table 5 and Table 6 .
It can be observed, that external loops are overpredicted, especially for the models trained on the differently distributed datasets 1-4. In contrast stacked bases are underpredicted. The percentage of bases in interior loop increases more drastically, with a longer sequence length for the prediction than for the ground truth, especially for models 0, 1 and 2 and model 1 trained on dataset 4. A similar effect is seen for multi loops, especially for model 1 trained on dataset 4. The poor predictions for the model trained on dataset 4 are expected, as this is the dataset consisting mainly of very short sequences.

In the second part the median, minimum and maximum lengths of different structural elements are analyzed and compared, see Table 7 and Table 8 .
The mean external loop lengths are larger for all predictions, than for the ground truth. The maximum external loop lengths are also larger for all predictions, especially for model 1 trained on dataset 1, 2 and 4, where there is always at least one sequence, which is completely unpaired.
The statistics for stacked bases are similar for the predictions and ground truth.
For multi loops both the median and maximum values for predictions are unreasonably large. For hairpin loops and interior loops the median values are similar to the ground truth, however there are also larger maximal values than in the ground truth.

Overall the results are in accordance to the other statistics, the overall topology is generally poorly predicted. The correct prediction of topology also needs a global overview of the structure, which seems to be more difficult for machine learning models to include.

Model	Test set	Bases (%)					
		External loops	Stacked	Bulges	Hairpin loops	Interior loops	Multi loops
Ground truth (RNAfold)	30	36.85	37.58	1.17	19.47	4.92	0.01
	40	28.36	43.06	1.90	18.44	8.19	0.04
	50	23.35	46.61	2.46	17.48	9.98	0.14
	60	20.27	48.98	2.89	16.41	10.85	0.59
	70	17.56	50.77	3.32	15.63	11.37	1.36
	80	15.40	52.09	3.34	15.02	12.15	2.00
	90	13.85	52.98	3.26	14.68	12.30	2.94
	100	12.32	54.12	3.13	14.29	12.61	3.52
Model 0	30	50.31	28.33	1.01	17.06	3.28	0.00
	40	40.15	34.64	1.83	16.79	6.54	0.04
	50	32.52	39.19	2.91	16.04	9.15	0.19
	60	28.20	40.96	4.00	15.39	10.73	0.73
	70	24.23	42.43	4.47	14.82	12.54	1.51
	80	21.96	42.03	4.78	15.45	13.40	2.38
	90	20.42	41.78	4.80	14.96	14.86	3.17
	100	18.07	41.60	4.88	15.09	15.82	4.55
Model 1	30	43.18	32.32	1.33	19.19	3.97	0.02
	40	34.52	37.55	2.26	18.28	7.36	0.04
	50	29.79	40.44	2.92	17.33	9.34	0.19
	60	27.14	41.65	3.59	15.83	11.12	0.66
	70	22.94	43.95	4.21	14.90	12.53	1.47
	80	22.15	42.63	4.47	14.89	13.73	2.13
	90	20.68	42.45	4.49	14.81	14.64	2.92
	100	18.81	42.11	4.46	14.67	15.66	4.29
Model 3	30	62.69	10.95	0.11	20.43	5.81	0.00
	40	47.71	23.35	0.52	19.20	9.21	0.00
	50	36.56	32.60	1.38	18.31	10.95	0.20
	60	27.67	38.74	2.05	17.36	13.36	0.80
	70	21.33	43.62	2.83	16.30	13.95	1.97
	80	19.96	41.96	3.33	15.77	15.80	3.18
	90	18.54	41.95	3.50	15.55	16.01	4.44
	100	17.06	41.45	3.47	14.56	17.93	5.54

Table 5.: The percentages of bases in different structural elements for the ground truth (which are the MFE structure predictions from RNAfold) and for Model 0, 1 and 3 trained on the dataset of 8000 sequences of length 70.
The results are based on tests sets of 2000 sequences for lengths 30 to 100.
Results are based on the upper triangular matrix with a threshold of 0.5 after the exclusion of multi-base pairs and pseudoknots.

Model	Test set	Bases (%)					
		External loops	Stacked	Bulges	Hairpin loops	Interior loops	Multi loops
Model 1 Dataset 1	30	44.93	31.74	0.92	18.72	3.68	0.00
	40	37.58	36.03	1.38	18.38	6.62	0.01
	50	33.14	38.75	1.86	17.37	8.78	0.11
	60	31.14	39.12	2.56	16.71	10.04	0.43
	70	28.87	39.46	3.25	15.53	11.58	1.31
	80	27.04	39.29	3.29	15.88	12.56	1.95
	90	25.78	38.72	3.26	15.80	13.42	3.03
	100	23.84	38.74	3.19	15.44	14.52	4.27
Model 1 Dataset 2	30	43.19	32.63	1.05	19.11	4.03	0.00
	40	34.84	37.92	1.83	18.16	7.23	0.01
	50	30.91	40.28	2.39	16.99	9.17	0.26
	60	29.14	40.83	2.56	15.81	11.00	0.66
	70	26.27	41.88	3.35	14.81	12.21	1.47
	80	23.74	42.36	3.45	14.62	13.75	2.08
	90	23.02	41.77	3.25	14.38	14.26	3.32
	100	21.05	42.00	3.14	14.04	15.02	4.75
Model 1 Dataset 3	30	48.52	29.93	0.93	17.13	3.48	0.00
	40	39.30	35.63	1.59	17.14	6.35	0.00
	50	34.35	38.77	2.32	15.70	8.77	0.10
	60	32.33	39.40	2.75	14.98	10.11	0.43
	70	29.39	40.15	3.58	14.35	11.56	0.96
	80	28.03	39.67	3.82	14.27	12.70	1.51
	90	26.65	39.23	3.75	14.35	13.11	2.91
	100	24.40	39.45	3.80	14.07	14.12	4.17
Model 1 Dataset 4	30	39.17	35.81	1.16	19.33	4.54	0.00
	40	31.55	40.49	1.90	18.61	7.32	0.12
	50	27.38	41.67	2.42	18.43	9.64	0.46
	60	26.55	40.68	2.72	18.01	10.53	1.52
	70	24.17	40.00	3.06	16.97	12.77	3.04
	80	22.33	38.93	3.03	17.79	13.25	4.67
	90	20.68	38.71	3.07	17.74	13.70	6.10
	100	18.97	38.63	2.84	17.65	14.25	7.65

Table 6.: The percentages of bases in different structural elements for Model 1 trained on datasets 1-4.
The results are based on tests sets of 2000 sequences for lengths 30 to 100.
Results are based on the upper triangular matrix with a threshold of 0.5 after the exclusion of multi-base pairs and pseudoknots.

Model	Test set	Median/Min-Max (length in bases)				
		External loops	Stacked	Hairpin loops	Interior loops	Multi loops
Ground truth (RNAfold)	30	10/1-30	5/3-21	6/2-20	3/2-13	3/3-3
	40	10/1-40	6/2-24	5/3-29	3/2-21	6/3-8
	50	10/1-50	6/2-24	5/3-35	3/2-25	6/2-12
	60	11/1-60	6/2-22	5/3-30	3/2-28	7/0-23
	70	11/1-49	6/2-30	5/3-33	3/2-28	7/0-24
	80	11/1-52	6/2-30	5/3-40	4/2-30	8/0-37
	90	11/1-66	8/2-28	5/3-43	4/2-30	9/0-37
	100	11/1-49	8/2-28	5/3-33	3/2-28	9/0-37
Model 0	30	13/1-30	6/2-20	6/3-24	3/2-20	2/2-2
	40	15/1-40	6/2-24	5/2-31	3/2-25	6/2-7
	50	15/1-50	6/2-24	5/3-34	3/2-27	7/1-20
	60	15/1-60	6/2-22	6/3-43	4/2-36	9/0-34
	70	15/1-60	6/2-26	6/3-54	4/2-37	10/0-30
	80	16/1-66	6/2-30	6/3-53	4/2-43	10/0-37
	90	16/1-67	6/2-26	6/3-49	4/2-43	11/0-40
	100	16/1-72	6/2-26	6/3-64	4/2-65	13/0-47
Model 1	30	11/1-30	6/2-20	6/3-26	3/2-15	10/10-10
	40	13/1-40	6/2-24	6/3-33	4/2-25	9/1-10
	50	13/1-50	6/2-24	6/3-39	4/2-27	8/2-19
	60	15/1-60	6/2-22	6/3-50	4/2-36	10/0-26
	70	14/1-56	6/2-26	6/3-44	4/2-45	12/1-34
	80	16/1-64	6/2-30	6/3-54	4/2-44	13/0-39
	90	17/1-63	6/2-28	6/3-66	4/2-55	13/0-54
	100	17/1-67	6/2-28	6/3-51	4/2-55	15/0-61
Model 3	30	18/1-30	2/2-18	9/3-26	3/2-19	x/x-x
	40	17/1-40	4/2-24	6/3-34	3/2-23	x/x-x
	50	16/1-50	6/2-24	6/3-44	4/2-35	13/4-22
	60	15/1-60	6/2-22	6/3-49	4/2-36	14/1-30
	70	13/1-60	6/2-26	6/3-54	4/2-47	12/0-38
	80	14/1-62	6/2-30	6/3-47	4/2-53	14/0-44
	90	15/1-67	6/2-26	6/3-58	4/2-57	15/0-59
	100	15/1-75	6/2-26	6/3-49	4/2-48	16/0-58

Table 7.: The median, minimum and maximum length of different structural elements for the ground truth (which are the MFE structure predictions from RNAfold) and for Model 0, 1 and 3 trained on the dataset of 8000 sequences of length 70.
The lengths of interior loops and multi loops are based on the individual parts of the loops.
The results are based on tests sets of 2000 sequences for lengths 30 to 100.
Results are based on the upper triangular matrix with a threshold of 0.5 after the exclusion of multi-base pairs and pseudoknots.

Model	Test set	Median/Min-Max (length in bases)				
		External loops	Stacked	Hairpin loops	Interior loops	Multi loops
Model 1 Dataset 1	30	12/1-30	6/2-20	6/3-26	2/2-16	x/x-x
	40	14/1-40	6/2-24	6/3-34	3/2-25	12/12-12
	50	15/1-50	6/2-24	6/3-41	3/2-30	7/1-19
	60	17/1-60	6/2-22	6/3-50	4/2-32	13/0-22
	70	18/1-70	6/2-26	6/3-55	4/2-42	13/0-40
	80	19/1-80	6/2-30	6/3-63	4/2-61	15/1-41
	90	21/1-90	6/2-28	6/3-75	4/2-70	18/1-52
	100	20/1-100	6/2-28	6/3-87	4/2-75	19/0-56
Model 1 Dataset 2	30	11/1-30	6/2-20	6/3-25	2/2-11	x/x-x
	40	13/1-40	6/2-24	6/3-29	3/2-21	9/9-9
	50	14/1-50	6/2-24	6/3-38	3/2-29	9/3-20
	60	16/1-60	6/2-22	6/3-45	4/2-38	10/0-24
	70	17/1-62	6/2-24	6/3-41	4/2-47	12/0-33
	80	17/1-64	6/2-30	6/3-57	4/2-46	14/0-47
	90	19/1-80	6/2-28	6/3-56	4/2-47	16/0-50
	100	19/1-87	6/2-28	6/3-50	4/2-62	17/0-58
Model 1 Dataset 3	30	13/1-30	6/2-20	6/3-26	2/2-18	x/x-x
	40	15/1-40	6/2-24	6/3-29	3/2-21	x/x-x
	50	16/1-50	6/2-24	6/3-36	3/2-27	10/1-25
	60	18/1-60	6/2-22	6/3-43	4/2-34	11/0-26
	70	19/1-70	6/2-26	6/3-52	4/2-44	12/0-36
	80	20/1-80	6/2-30	6/3-56	4/2-53	15/0-43
	90	21/1-90	6/2-28	6/3-70	4/2-56	17/0-59
	100	21/1-100	6/2-28	6/3-62	4/2-75	19/0-64
Model 1 Dataset 4	30	11/1-30	6/2-20	5/3-21	3/2-12	x/x-x
	40	12/1-40	6/2-24	5/3-30	3/2-24	7/2-14
	50	12/1-50	6/2-24	6/3-43	4/2-35	10/0-21
	60	14/1-60	6/2-22	6/3-49	4/2-43	12/0-32
	70	15/1-70	6/2-24	6/3-56	4/2-49	14/0-37
	80	16/1-80	6/2-30	6/3-68	4/2-55	16/0-47
	90	16/1-78	6/2-24	6/3-83	4/2-64	17/0-56
	100	16/1-100	6/2-26	6/3-65	4/2-72	18/0-71

Table 8.: The median, minimum and maximum length of different structural elements for Model 1 trained on datasets 1-4.
The lengths of interior loops and multi loops are based on the individual parts of the loops.
The results are based on tests sets of 2000 sequences for lengths 30 to 100.
Results are based on the upper triangular matrix with a threshold of 0.5 after the exclusion of multi-base pairs and pseudoknots.

Part IV.

Conclusion and outlook

The first part of this thesis is based on the idea of implementing a simple baseline approach for RNA structure prediction, were the basic idea is to simplify the problem by predicting unpaired and paired regions instead of full secondary structures.
This has the advantage of avoiding post processing steps or constraints during training which are needed to enforce the formation of feasible secondary structures.
However, the poor performances seen with this methods indicate that the simplification of secondary structure by paired and unpaired regions is not suitable, in contrast to the simplification secondary structure offers for the full tertiary structure.

This leads to the need for other representation methods of the full RNA secondary structure, which are also compatible with machine learning approaches.
Two, already published methods, E2Efold [Chen et al., 2020b] and SPOT-RNA [Singh et al., 2019], rely on the representation of structures as matrices, with different approaches to ensure the prediction of secondary structures which fulfill the general constraints. For SPOT-RNA constraints are learned from data, while E2Efold enforces the constraints through iterative optimization.

One of them, SPOT-RNA, a ResNet architecture, coupled with either a FC block or a 2D-BLSTM, has been reimplemented and tested.
While this approach reaches good performances with MCC values up to 0.7-0.8, the results, such as performance on different datasets and the evaluation on secondary structures constrains also highlights the shortcomings models may have in their generalization ability

As discussed in chapter 6.1. the choice of datasets is a challenging part for machine learning as overfitting may falsely inflate the reported performances.
Furthermore it has been shown that differences on the sequence level are not enough to detect overfitting and tests with structurally different RNAs need to be performed [Rivas et al., 2012, Rivas, 2013].
Currently methods mainly remove too similar sequences from datasets using sequence similarity measures [Chen et al., 2020b, Singh et al., 2019], with only little discussion about biases in datasets and the effect of too structurally similar datasets.
Overall the choice of training, validation and test data seems to be done poorly, without respect to the particular challenges for RNA dataset, such as structural dissimilarity and length distributions. This indicates that the high performances reported seem to be a least partially due to overfitting.
In this thesis the effect different datasets can have on the generalization ability is demonstrated on the example of different sequence length. This effect can also be amplified by the choice of model architecture.

While more tests with different architectures and broader sequence length ranges are needed, the results already highlight the need for a discussion about generalization abilities, as well as consistency in evaluation of different machine learning approaches and a clear communication of chosen datasets and more extensive criteria for divisions into training, validation and test set.

Especially as validation and also test datasets often have the same or similar distributions as the training set, a possible lack in generalization ability can remain undetected during model evaluation.
Possible similarities in datasets are for example the already mentioned distribution of sequence lengths as well as the types of RNAs. The choice of datasets could be based on the approach described in [Rivas et al., 2012] ensuring sequences dissimilarity in sequences as well as structures.

It could also be further explored, if the performance for different sequence lengths are more dependant on the length of sequences in the training set or on the general trend that shorter sequences are easier to predict. Both effects can be seen in this thesis to some extend.

This thesis also highlights that evaluations beyond the basic performance metrics, such as MCC, can give more detailed insights into the capabilities of machine learning models. The incorporation of global properties seems to be especially challenging and offers the possibilities for further research.
This inability to properly represent global properties also challenges some of the often advertised advantages for machine learning, such as the prediction of pseudoknots and the inclusion of structural elements such as triple base pairs.
As models already show inabilities to represent global properties, when enough data is offered, such as for example for the number of base pairs in respect to sequence length, this is especially challenging for the previously mentioned features for which datasets are generally sparse.

Additional research on how it could be possible to ensure the known global constraints for RNA secondary structures are fulfilled is needed, as both learning them from data and iterative optimization can not currently guarantee this. For the iterative optimization method used in E2Efold [Chen et al., 2020b] one solution can be to learn when to stop the algorithm while training the model [Chen et al., 2020a].
Another solution could be to use the machine learning prediction as basis and apply post processing steps, such as a Nussinov-type/dynamic programming algorithm.

Research should not be limited to enforcing known constraints as one possible application of machine learning approaches could be to include long distance interactions, which are not captured by dynamic programming approaches. This need a good understanding of the way in which machine learning models learn and represent global structural data beyond the development of new models which report better overall performances.
Generally the inclusion of more information as input for the models could provide benefits, for example including including position information for matrices, which is already done in [Chen et al., 2020b].

Bibliography

[Andronescu et al., 2014] Andronescu, M., Condon, A., Turner, D., and Mathews, D. (2014). The determination of rna folding nearest neighbor parameters. *Methods in molecular biology (Clifton, N.J.)*, 1097:45–70.

[Angioloni, 2019] Angioloni, L. (2019). Proteinsecondarystructure-cnn. https://github.com/LucaAngioloni/ProteinSecondaryStructure-CNN. Accessed: January 05, 2021. Commit: 69865a60ee2df30cd8c3439aa9b4b22d8f94cb47.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166.

[Brion and Westhof, 1997] Brion, P. and Westhof, E. (1997). Hierarchy and dynamics of rna folding. *Annual Review of Biophysics and Biomolecular Structure*, 26(1):113–137.

[Chen et al., 2020a] Chen, X., Dai, H., Li, Y., Gao, X., and Song, L. (2020a). Learning to stop while learning to predict.

[Chen et al., 2020b] Chen, X., Li, Y., Umarov, R., Gao, X., and Song, L. (2020b). Rna secondary structure prediction by learning unrolled algorithms.

[Chicco and Jurman, 2020] Chicco, D. and Jurman, G. (2020). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21.

[Danaee et al., 2018] Danaee, P., Rouches, M., Wiley, M., Deng, D., Huang, L., and Hendrix, D. (2018). Bprna: Large-scale automated annotation and analysis of rna secondary structure. *Nucleic acids research*, 46.

[Do et al., 2006] Do, C., Woods, D., and Batzoglou, S. (2006). Contrafold. *Bioinformatics*, 22:e90–e98.

[Dowell and Eddy, 2004] Dowell, R. and Eddy, S. (2004). Evaluation of several light-weight stochastic context-free grammars for rna secondary structure prediction. *BMC bioinformatics*, 5:71.

[Gautheret et al., 1995] Gautheret, D., Damberger, S. H., and Gutell, R. R. (1995). Identification of base-triples in rna using comparative sequence analysis. *Journal of Molecular Biology*, 248(1):27 – 43.

[Gilbert, 1986] Gilbert, W. (1986). Origin of life: The rna world. *Nature*, 319(618).

[Graves et al., 2007] Graves, A., Fernández, S., and Schmidhuber, J. (2007). Multi-dimensional recurrent neural networks. *CoRR*, abs/0705.2011.

[Greff et al., 2015] Greff, K., Srivastava, R., Koutník, J., Steunebrink, B., and Schmidhuber, J. (2015). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28.

[Hajdin et al., 2013] Hajdin, C., Bellaousov, S., Huggins, W., Leonard, C., Mathews, D., and Weeks, K. (2013). Accurate shape-directed rna secondary structure modeling, including pseudoknots. *Proceedings of the National Academy of Sciences of the United States of America*, 110.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. pages 770–778.

[Hermann and Westhof, 1999] Hermann, T. and Westhof, E. (1999). Non-watson-crick base pairs in rna-protein recognition. *Chemistry & Biology*, 6(12):R335 – R343.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[Jones and Ferré-D'Amaré, 2015] Jones, C. P. and Ferré-D'Amaré, A. R. (2015). Rna quaternary structure and global symmetry. *Trends in biochemical sciences*, 40(4):211–220.

[Joyce, 1989] Joyce, G. (1989). Rna evolution and the origins of life. *Nature*, 338:217–224.

[Joyce, 2002] Joyce, G. (2002). The antiquity of rna-based evolution. *Nature*, 418:214–221.

[Lorenz et al., 2011] Lorenz, R., Bernhart, S., Höner zu Siederdissen, C., Ta, H., Flamm, C., Stadler, P., and Hofacker, I. (2011). Viennarna package 2.0. *Algorithms for molecular biology : AMB*, 6:26.

[Mathews and Turner, 2006] Mathews, D. and Turner, D. (2006). Prediction of rna secondary structure by free energy minimization. *Current Opinion in Structural Biology*, 16(3):270–278. Nucleic acids/Sequences and topology.

[Mathews et al., 2004] Mathews, D. H., Disney, M. D., Childs, J. L., Schroeder, S. J., Zuker, M., and Turner, D. H. (2004). Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of rna secondary structure. *Proceedings of the National Academy of Sciences*, 101(19):7287–7292.

[Mathews et al., 1999] Mathews, D. H., Sabina, J., Zuker, M., and Turner, D. H. (1999). Expanded sequence dependence of thermodynamic parameters improves prediction of rna secondary structure11edited by i. tinoco. *Journal of Molecular Biology*, 288(5):911 – 940.

[Mattick, 2001] Mattick, J. S. (2001). Non-coding rnas: the architects of eukaryotic complexity. *EMBO reports*, 2(11):986–991.

[Mattick, 2003] Mattick, J. S. (2003). Challenging the dogma: the hidden layer of non-protein-coding rnas in complex organisms. *BioEssays*, 25(10):930–939.

[Nussinov and Jacobson, 1980] Nussinov, R. and Jacobson, A. B. (1980). Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313.

[Rivas, 2013] Rivas, E. (2013). The four ingredients of single-sequence rna secondary structure prediction. a unifying perspective. *RNA Biology*, 10:1185 – 1196.

[Rivas et al., 2012] Rivas, E., Lang, R. W., and Eddy, S. (2012). A range of complex probabilistic models for rna secondary structure prediction that includes the nearest-neighbor model and more. *RNA*, 18 2:193–212.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

[Rumelhart et al., 1986] Rumelhart, D., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

[Singh et al., 2019] Singh, J., Hanson, J., Paliwal, K., and Zhou, Y. (2019). Rna secondary structure prediction using an ensemble of two-dimensional deep neural networks and transfer learning. *Nature Communications*, 10.

[Sloma and Mathews, 2016] Sloma, M. and Mathews, D. (2016). Exact calculation of loop formation probability identifies folding motifs in rna secondary structures. *RNA*, 22.

[Staple and Butcher, 2005] Staple, D. and Butcher, S. (2005). Pseudoknots: Rna structures with diverse functions. *PLoS biology*, 3:e213.

[Stombaugh et al., 2009] Stombaugh, J., Zirbel, C., Westhof, E., and Leontis, N. (2009). Frequency and isostericity of rna base pairs. *Nucleic acids research*, 37:2294–312.

[Tan et al., 2017] Tan, Z., Fu, Y., Sharma, G., and Mathews, D. (2017). Turbofold ii: Rna structural alignment and secondary structure prediction informed by multiple homologs. *Nucleic acids research*, 45.

[Varani and McClain, 2000] Varani, G. and McClain, W. H. (2000). The g x u wobble base pair. a fundamental building block of rna structure crucial to rna function in diverse biological systems. *EMBO reports*, 1(1):18–23.

[Visin et al., 2015] Visin, F., Kastner, K., Cho, K., Matteucci, M., Courville, A. C., and Bengio, Y. (2015). Renet: A recurrent neural network based alternative to convolutional networks. *CoRR*, abs/1505.00393.

[Waterman and Smith, 1978] Waterman, M. and Smith, T. (1978). Rna secondary structure: a complete mathematical analysis. *Bellman Prize in Mathematical Biosciences*, 42:257–266.

[Zakov et al., 2011] Zakov, S., Goldberg, Y., Elhadad, M., and Ziv-Ukelson, M. (2011). Rich parameterization improves rna structure prediction. *Research in Computational Molecular Biology*, 6577.

[Zhang et al., 2019] Zhang, H., Zhang, C., Li, Z., Li, C., Wei, X., Zhang, B., and Liu, Y. (2019). A new method of rna secondary structure prediction based on convolutional neural network and dynamic programming. *Frontiers in Genetics*, 10.

[Zhao et al., 2018] Zhao, Y., Wang, J., Zeng, C., and Xiao, Y. (2018). Evaluation of rna secondary structure prediction for both base-pairing and topology. *Biophysics Reports*, 4.